Nr.02 | Frühling 2012 | www.ijug.eu

Java aktuell

Das Magazin der Java-Community

Java aktuell

Java im Aufwind

VisualVM
Unbekannte Kostbarkeiten des SDK

Grails
Die Suche ist vorbei

Microsoft und Java
Frei verfügbare Angebote für Software-Entwickler

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

4 191978 304903 02

iJUG Verbund

Wolfgang Taschner
Chefredakteur Java aktuell

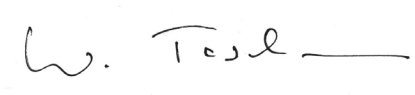
Zahlen sind das eine …

Oracle glänzt gerne mit Zahlen. Soundso viel Milliarden Geräte laufen mit Java, soundso viel Millionen Downloads von Java, soundso viel Millionen Java-Entwickler gibt es weltweit. Das kann sich sehen lassen. Doch wichtiger, als sich auf Zahlen auszuruhen, ist für mich der Blick nach vorn. In welche Richtung soll sich Java weiterentwickeln, wo sind Lücken, wo muss nachgebessert werden und vor allem: Wo sind die innovativen Bereiche, die die Zukunft von Java sichern.

Gerade in Bezug auf die Zukunft von Java ist auch der Interessenverbund der Java User Groups e.V. (iJUG), Herausgeber dieser Zeitschrift, aktiv geworden. Der iJUG ist seit diesem Jahr Mitglied im Java Community Process – dem Gremium, das über die Weiterentwicklung von Java bestimmt. Ziel des fünfköpfigen iJUG-Teams ist die umfassende Vertretung der gemeinsamen Interessen der deutschsprachigen Java-Usergroups im JCP. Ein besonderes Augenmerk liegt auf der Mitarbeit am JSR 308 (Annotations), am JSR 335 (Lambda) sowie am JSR zur Java Platform Enterprise Edition (siehe Seite 21).

Wir freuen uns auf Ihre Wünsche und Anregungen sowohl zum JCP als auch zu dieser Ausgabe der Java aktuell.

Ihr



Interview mit John Duimovich, Distinguished Engineer in der IBM Software Group mit Schwerpunkt „Java Virtual Machines & Embedded Java", IBM Canada, Seite 16

„Java besitzt immer noch ein enormes Potenzial …" Stefan Koospal, Vorsitzender der Sun User Group Deutschland, im Gespräch mit Java aktuell, Seite 42

Der Sport ist in manchen Bereichen hilfreich für die Entwicklung von Software. Was wir vom America's Cup lernen können, Seite 44

Das Java-Tagebuch

Das Java-Tagebuch gibt einen Überblick über die wichtigsten Geschehnisse rund um Java – in komprimierter Form und chronologisch geordnet. Der vorliegende Teil widmet sich den Ereignissen im vierten Quartal 2011.

4. Oktober 2011

Apache TomEE ist „Java EE 6 Web Profile"-zertifiziert

Die Apache Software Foundation verkündet, dass der Application-Server TomEE (sprich: „Tommy") die Kompatibilitäts-Tests für das Java EE 6 Web Profile bestanden hat – eine leichtgewichtigere Version von Java EE. Das TomEE-Projekt selbst ist noch recht jung, setzt aber auf anderen bewährten Apache-Projekten wie Tomcat, MyFaces und OpenEJB auf und mixt daraus ein neues Produkt.

10. Oktober 2011

Google Dart – ein weiterer Sargnagel für „mobile Java"?

Google hat auf der „GOTO"-Konferenz in Aarhus Details zu seiner neuen Web-Programmiersprache „Dart" bekanntgegeben. Sie soll angeblich letztlich JavaScript als „die" Web-Programmiersprache ablösen – wenn man einer Google-internen Mail glauben darf, die an die Öffentlichkeit gelangte und über den englischen Wikipedia-Eintrag zu „Dart (programming language)" einzusehen ist. Die Google-Vertreter in Aarhus bestreiten dies allerdings und geben stattdessen „fragmented mobile platforms" als den eigentlichen „Gegner" an (siehe http://dartinside.com/2011/live-from-dart-launch/).

Darf man als künftiges Einsatzgebiet also doch eher Android als das GWT ansehen? Die offiziellen „Design Goals" von Dart lassen diese Option zu, eines lautet: „Make Dart appropriate for the full range of devices on the web – including phones, tablets, laptops, and servers." Es wäre sicher ein hartes Stück Arbeit, könnte Google aber als Alternative zum Pseudo-Java vor dem Super-GAU im Rechtsstreit mit Oracle bewahren. Aber für die Optimisten unter uns, die immer noch an eine einheitliche „mobile Java"-Plattform glauben (in welcher Form auch immer), wäre das sicher kein Grund zur Freude: Dart wird nicht in (Java) Byte Code umgewandelt, sondern in einer eigenen Dart VM interpretiert. Die obige Aussage über „fragmented mobile platforms" könnte man dann so verstehen, dass Dart (beziehungsweise Android) die Zukunft der mobilen Plattformen sein soll.

19. Oktober 2011

RIM verzichtet auf Java in neuer BBX-Plattform

Stichwort „Fragmentierung der mobilen Plattformen": Research In Motion hat bekanntgegeben, dass das neue Betriebssystem BlackBerry BBX (inzwischen umbenannt in „BlackBerry 10") kein Java SDK mehr enthalten wird. Stattdessen sollen das „Native SDK" (C/C++) und das „WebWorks SDK" (JavaScript, HTML 5, CSS) unterstützt werden. Der Hersteller wird aber nicht müde zu betonen, dass damit Java auf dem BlackBerry noch lange nicht tot ist. Es seien aktuell mehr als 70 Mio. BlackBerry-Smartphones auf dem Markt und die Zahl dürfe noch wachsen, da die ersten BlackBerry-7-Geräte gerade erst ausgeliefert wurden. Das heißt, Java ME als Umsatzgarant für Oracle dürfte noch eine geraume Zeit weiter existieren. Aber eine rosige Perspektive ist das nicht.

22. Oktober 2011

Populäre Programmiersprachen: Java und C Kopf an Kopf

Kevin Farnham beschäftigt sich in seinem Blog auf „weblogs.java.net" mit einem InfoWorld-Artikel unter dem zurückhaltenden Titel „Java losing popularity among developers". Der Artikel basiert auf der Oktober-Ausgabe des TIOBE-Index (siehe http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html), der die Popularität von Programmiersprachen anhand von Internet-Such-Resultaten misst. Dieser Index zeigt einen leicht fallenden Trend für Java und einen leicht steigenden für C (Letzteres vielleicht aufgrund steigender Popularität von SDKs wie dem oben erwähnten von BlackBerry – so, wie Apple Objective-C aus der Vergessenheit zurückgeholt hat?). Letztlich betreibt Farnham ein paar statistische Überlegungen und analysiert die Berechnungsmethoden für den TIOBE-Index mit dem Fazit, dass sich kein Java-Entwickler Sorgen um die Zukunft von Java oder gar seine eigene machen muss. Soweit ganz interessant, aber nichts, was nicht jeder ohnehin geahnt hätte.

Vielleicht noch interessanter ist ein Blick in die Details des Index. Dort wird Java mit gut siebzehn Prozent an Platz eins gelistet, 2004 waren es noch teilweise fünfundzwanzig Prozent (alle Programmiersprachen zusammen kommen auf hundert Prozent). Jetzt könnte man vermuten, dass Java unter anderem deswegen an Popularität verliert, weil andere Programmiersprachen auf der JVM genutzt werden. Das ist sicher ein Faktor, aber es ist doch erstaunlich, dass Sprachen wie Clojure und Scala, die momentan einen Hype erfahren, jeweils noch nicht einmal auf 0,25 Prozent kommen und es damit nicht in die Top 50 schaffen. Aber, wie Kevin Farnham schon argumentiert hat, der Index ist auch nur eine Weise, die Welt der Programmiersprachen statistisch messbar zu machen, und repräsentiert nicht notwendigerweise akkurat, was gerade tatsächlich passiert.

26. Oktober 2011

Java 7 Update 1: JIT-Compiler-Fehler behoben

Die JIT-Compiler-Fehler in Java 7, die zu Abstürzen von Apache Lucene und Solr durch fehlerhafte Schleifen-Optimierungen führten, sind mit Java 7 Update 1 behoben. Nachdem die entsprechenden

Bugs zunächst nicht in den Release-Notes aufgeführt waren, hat Oracle dies nun nachgeholt – und Lucene-Committer Uwe Schindler hat es bereits verifiziert.

1. November 2011

JCP 2011 Executive Committee Election Under Way; „Meet the Candidates"-Call on Thursday

In der Öffentlichkeit, aber dennoch geräuschloser als beim letzten Mal, haben Neuwahlen für die Executive Committees (EC) des Java Community Process stattgefunden. Bei den Wahlen zum EC für Java SE/EE haben Ericsson, Intel und SAP ihre ratifizierten Sitze wiedergewonnen (von Oracle vorgeschlagene, nur zu bestätigende Mitglieder). Azul Systems und Twitter haben die frei gewählten Sitze gewonnen und ersetzen damit VMWare und Werner Keil. Während für die drei Kandidaten zu den ratifizierten Sitzen das Risiko nur in einer mehrheitlichen Ablehnung bestand, was 2010 aber schon einmal eingetreten ist, herrschte bei den offenen Wahlen (erfreulich) viel Konkurrenz.

Bei den Wahlen zum EC für Java ME war das Bild ähnlich. Hier gingen die drei ratifizierten Sitze wieder an IBM und Nokia sowie als Neuling SK Telecom (für AT&T), die frei gewählten Sitze besetzen nun ARM Limited und Werner Keil anstelle von Sean Sheedy und Alex Terrazas.

Alle Kandidaten haben sich gemäß den Zielen der JCP-Erneuerung am 20. Oktober 2011 in einem öffentlichen Conference Call („Meet the Candidates") vorgestellt und Fragen beantwortet.

Oracle beantragt Aufnahme von JavaFX in das OpenJDK

Wie auf der JavaOne 2011 angekündigt, wird JavaFX Open Source – zumindest wenn der jetzt von Oracle-Vertreterin Iris Clark eingereichte „call for votes" vom OpenJDK-Projekt akzeptiert wird. Unter dem Namen „OpenJFX" soll es dann als Projekt innerhalb des OpenJDK weitergeführt werden. Oracle will die Sourcen dazu schrittweise im Laufe der nächsten Monate übergeben, am Ende soll es dann keine Abhängigkeiten mehr von Oracle Binaries geben. Bis zum 16. November 2011 können OpenJDK Members nun über den Vorschlag abstimmen.

4. November 2011

Java User Groups sollen JSRs „adoptieren"

Aus der London Java Community – einer der beiden im Executive Committee des JCP vertretenen JUGs – kommt der Vorschlag zu einem „adopt a JSR"-Programm. Dieses soll Java User Groups motivieren, sich einen Java Specification Request (oder mehrere) zu wählen und sich darum zu kümmern. Die JUGs sollen direkten Kontakt mit dem Specification Lead aufnehmen und die Community in den JSR einbinden, bzw. auch umgekehrt den JSR in der Community bekannt machen.

Dies ist eine großartige Idee, um die aktive Community weiter in die Entwicklung der Zukunft von Java einzubinden. Mehrere Java User Groups haben sich der Initiative inzwischen angeschlossen, der iJUG hat ebenfalls im November eine Beteiligung beschlossen (siehe Seite 20).

13. November 2011

Apache Geronimo vollständig für Java EE 6 zertifiziert

Als nächster Application-Server ist Apache Geronimo als hundert Prozent kompatibel mit Java EE 6 zertifiziert worden. Im Sommer war dies bereits für WebSphere erfolgt. Von den „großen" Servern fehlen damit nur noch JBoss, der aber immerhin bereits das Java-EE-6-Web-Profile unterstützt, und WebLogic, der jedoch in Kürze in der Version 12c ebenfalls den Stempel tragen soll.

16. November 2011

OpenJFX ist ins OpenJDK aufgenommen

Wie zu erwarten, ist OpenJFX als Projekt in das OpenJDK aufgenommen worden. Die Beteiligung unter den abstimmungsberechtigen „OpenJDK Members" war mit zwölf abgegebenen Stimmen recht gering, es gab allerdings nur „Ja"-Stimmen.

Apache Harmony verschwindet auf dem Dachboden

Ein guter Tag für das OpenJDK, ein schlechter für Apache Harmony, das auch einmal angetreten war, der Welt ein Open Source Java SE zur Verfügung zu stellen. Nachdem IBM Ende 2010 seine Aktivitäten zum OpenJDK verlagert hatte, fehlte wohl die Unterstützung oder der Glaube, dass es daneben für Harmony noch eine Zukunft gibt. Harmonys „Project Management Committee" hat sich daher nahezu einstimmig dafür ausgesprochen, das Projekt in den Attic („Dachboden") von Apache zu verschieben, in dem alle Projekte landen, die nicht mehr aktiv weitergeführt werden.

18. November 2011

DOAG 2011 und Devoxx

Diese Woche finden, wie auch in den Vorjahren, zeitgleich die Devoxx (Konferenz der belgischen Java User Group in Antwerpen) und die DOAG 2011 Konferenz + Ausstellung (DOAG Deutsche ORACLE-Anwendergruppe e.V. in Nürnberg) statt. Während die Devoxx inzwischen „die" Java-Konferenz in Europa ist, liegt der Fokus der DOAG-Konferenz auf allen Themen rund um Oracle und damit auch (aber nicht nur) auf Java. Die Devoxx zieht quasi schon traditionell viele hochkarätige Redner und eine große, aktive Community an. Bei der DOAG steht das Thema „Java" immer noch im Schatten der Datenbank. Aber es wird besser, was einige klangvolle Namen unter den Referenten zeigen: Ed Burns (Specification Lead für Java Server Faces), Terrence Barr („Java Mobile und Embedded"-Guru bei Sun/Oracle) und Dalibor Topic (Java F/OSS Ambassador bei Sun/Oracle und ein Lead in der OpenJDK Community) kehren sicher nicht grundlos zurück. Neu dabei war diesmal auch Danny Coward (Principal Engineer bei Oracle, bereits bei Sun Chief Architect für Client Software und Repräsentant im JCP).

Wie es aussieht, lässt sich die Termin-Kollision der beiden Konferenzen nicht so einfach auflösen. Zur Vorbeugung einer Optionsparalyse hat Markus Eisele in seinem Blog eine gute Entscheidungshilfe für Konferenzbesucher gegeben (siehe http://blog.eisele.net/2011/11/doag-2011-vs-devoxx-value-and.html).

Vielleicht gelingt es 2012 sogar, aus dem Konflikt einen Vorteil zu generieren und noch mehr renommierte Sprecher aus den USA mit der Möglichkeit eines Auftritts auf zwei Konferenzen anzulocken.

1. Dezember 2011

WebLogic Server 12c ist verfügbar

Oracle gibt den WebLogic Server in der Version 12c (c wie „cloud") frei. Der Server ist hundert Prozent kompatibel mit Java EE 6. Im Bereich des Bauens und Deployens von Applikationen hat Oracle viele Dinge verbessert, darunter die Maven-3-Unterstützung. Aber auch das „Verständnis" für GlassFish-Deployment-Deskriptoren wurde weiterentwickelt, um eine einfachere Migration von GlassFish auf das High-End-Produkt zu ermöglichen.

11. Dezember 2011

„java.net"-Umfrage: „Warum erhält Desktop-Entwicklung heute so wenig Aufmerksamkeit?"

Redaktionsleiter Kevin Farnham präsentiert die Ergebnisse der letzten Umfrage: Warum erhält Desktop-Entwicklung heute so wenig Aufmerksamkeit? Hier die (wie immer vorgegebene) Liste von Antworten in der Reihenfolge der erhaltenen Stimmen:

1. Der Desktop ist auf kleinere, mobile Geräte weitergezogen, wo er viel Aufmerksamkeit erhält. 30 Prozent (143 Stimmen)
2. Weil Desktop-Entwickler ihre Ergebnisse nicht angemessen publizieren. 16 Prozent (76 Stimmen)
3. Weil heute fast keine Desktop-Entwicklung mehr passiert. 15 Prozent (74 Stimmen)
4. Ich weiß nicht. 12 Prozent (57 Stimmen)
5. Anderer Grund. 12 Prozent (57 Stimmen)
6. Weil Sites wie java.net den Desktop ignorieren. 9 Prozent (42 Stimmen)
7. Weil Desktop-Entwicklung und Apps uninteressant sind. 7 Prozent (32 Stimmen)

Die „Gewinner-Antwort" kommt sicher nicht überraschend, und Farnham wirft die Frage auf, ob man die heutigen Pads und Smartphones nicht eher als „kleine Desktops" bezeichnen sollte. Zu Antwort Nummer zwei verweist er auf den Unterschied zwischen Technologien, die eingesetzt werden, und solchen, die auf Konferenzen, in Artikeln etc. präsentiert werden. Vielleicht geschieht Desktop-Entwicklung im Stillen, ohne dass viel darüber berichtet wird. Farnham verbindet dies zusammen mit Antwort Nummer sechs zu einer Aufforderung an Desktop-Entwickler, mehr auf java.net und anderen Plattformen zu publizieren.

Sicher, die typischen Desktop-Anwendungen haben ihre Hype-Zeiten lange hinter sich und werden heute häufig auch dort durch Web-Technologien ersetzt, wo es unter dem Gesichtspunkt der Anwenderfreundlichkeit keinen Sinn ergibt. Technologien wie JavaFX, das ja zuerst mal auf Desktop-Entwicklung ausgerichtet wurde, könnten eigentlich ein bisschen Aufmerksamkeit zurückbringen. Aber selbst von der JavaOne, auf der es viele Sessions zu JavaFX gab, sind eher diejenigen in Erinnerung geblieben, die die Lauffähigkeit der neuen Technologie auch auf dem iPad und Android demonstrierten. Vermutlich ist der „kleine Desktop" als jüngere Technologie mit starkem Wachstum einfach attraktiver.

16. Dezember 2011

Oracle Java verschwindet aus Linux-Distributionen

Nachdem Oracle bereits Ende August die „Operating System Distributor License for Java" aufgekündigt hat, hagelt es weiterhin Proteste gegen diese Entscheidung. Die seit 2006 von Sun vergebene Lizenz für Linux- und Solaris-Distributoren erlaubte diesen die Paketierung und (automatische) Verteilung des Sun JRE und JDK inklusive Online-Updates mit ihren Distributionen. Oracle hatte diese Lizenz weitergeführt. Nun aber sieht der Konzern keine Notwendigkeit mehr, da das OpenJDK weit genug gediehen sei. Das Oracle JRE und JDK könne weiterhin von der Oracle-Website heruntergeladen werden.

Am 14. und 15. Dezember 2011 haben nun Debian und Ubuntu verkündet, dass aufgrund der gekündigten Lizenz nicht nur keine Updates mehr erfolgen, sondern die Pakete aus Sicherheitsgründen sogar entfernt werden – das heißt, mit dem nächsten Update wird das Oracle JRE beziehungsweise JDK von der Maschine gelöscht. Sie empfehlen (ebenso wie Oracle) den Umstieg auf das OpenJDK.

Da es aber immer noch Software gibt, die mit dem Oracle JRE, aber nicht mit dem OpenJDK läuft (Version 6), manuelle Updates aber keine wirkliche Alternative sind, lassen viele System-Administratoren ihrem Ärger in den Blogs freien Lauf. Die Aufkündigung der Lizenz sei einfach zu früh gekommen, lauten die noch milderen Kommentare. Einige Blogger sehen darin allerdings auch etwas Positives: Nämlich dass nun alle gezwungen sind, das OpenJDK zu nutzen (und Bugs zu melden statt das Oracle JRE zu installieren), was langfristig dem OpenJDK zugutekommt.

17. Dezember 2011

Java 7 Update 2: Jetzt mit JavaFX SDK

Das zweite Update von Java 7 SE enthält jetzt auch das JavaFX SDK. Daneben unterstützt es Solaris 11 und Firefox in den Versionen 5, 6, 7 und 8.

19. Dezember 2011

„-Xmx" = „-Xms"?

Kirk Pepperdine hat einen interessanten Blog-Eintrag unter „weblogs.java.net" geschrieben – zur alten Diskussion über Sinn und Unsinn des Gleichsetzens von Min Heap und Max Heap der JVM (die Start-Optionen -Xms und -Xmx). Diese Einstellung verhindert in heutigen JVMs das automatische Tunen der Young Generation und damit eine wesentliche Möglichkeit der Performance-Optimierung. Der immer noch nicht ganz produktionsreife G1 Garbage Collector soll, wenn es nach seinen Entwicklern geht, die Freiheit bekommen, die Young-Generation-Größe trotz fester Heap-Größe automatisch zu tunen. Pepperdine schreibt dazu sinngemäß: Wenn wir die Vorstellung der Leute vom Tuning der JVM nicht ändern können, ändern wir einfach die JVM, um dieser Vorstellung zu entsprechen …

21. Dezember 2011

Erstes Meilenstein-Release von Ceylon offiziell angekündigt

Chefentwickler Gavin King hat auf der Website „ceylon-lang.org" das erste offizielle Release von Ceylon angekündigt, das noch in derselben Woche zum Download bereitstehen soll: Ceylon M1 „Newton". Damit stehen der Compiler und Doc-Compiler, das grundlegende Sprachmodul und die Laufzeit-Umgebung zur Verfügung – mit einem Sprachumfang, der bereits weitestgehend dem von Java entsprechen soll, mit Ausnahme von Enumerations, nutzerdefinierten Annotations und Reflection.

In fünf solcher Meilenstein-Releases (M1 bis M5) wird die Entwicklung der Version 1.0 vollzogen. Die Inhalte der Meilensteine sind bereits geplant, M2 beschäftigt sich unter anderem mit Java-Interoperabilität und File I/O. Einen Zeitplan gibt es jedoch nicht, dieser soll flexibel gehalten werden, um vor einer Veröffentlichung jeweils die nötige Qualität zu garantieren.

27. Dezember 2011

„java.net"-Umfrage: „Was war wichtig in 2011, was wird in 2012 wichtig sein?"

Zum Jahreswechsel gibt es auf java.net zwei neue Umfragen: Was war 2011 das wichtigste Ereignis in Bezug auf Java/die JVM, und was sollte 2012 bringen?

Beim Rückblick auf 2011 votieren die meisten Teilnehmer (38 Prozent) für das Release von Java SE 7, mit weitem Abstand gefolgt vom „Wachstum von Android" (22 Prozent), der Freigabe von JavaFX als Open Source und der Verfügbarkeit auf Mac/Linux (13 Prozent) sowie der ausgearbeiteten Roadmap für Java SE 8 und 9 (11 Prozent). Alle anderen Antworten, unter anderem zum JCP und selbst zum Hype-Thema „Cloud Computing", rangieren unter „ferner liefen". Neben Absichtserklärungen und Roadmaps gab es zur Cloud 2011 allerdings auch nichts wirklich Greifbares, auf das sich die Teilnehmer hätten beziehen können.

Die Umfrage, worauf sich die wichtigsten Ereignisse 2012 beziehen werden, folgt direkt hinterher. Hier entfallen die meisten Stimmen auf „Android" (22 Prozent), gefolgt von „Java SE/OpenJDK" (18 Prozent) und Java EE (17 Prozent). Trotz – oder gerade wegen – des anhaltenden Rechtsstreits zwischen Google und Oracle ist das Thema „Android" bei den Java-Entwicklern also weiterhin hoch im Rennen – denn die Vermutung liegt nahe, dass die meisten Teilnehmer an den „java.net"-Umfragen Entwickler sind oder zumindest im Umfeld der Software-Entwicklung tätig sind.

6. Januar 2012

NetBeans 7.1 freigegeben – mit JavaFX 2.0

Oracle hat die neue Version 7.1 der Entwicklungsumgebung NetBeans freigegeben und dürfte damit insbesondere den Anhängern gehobener Oberflächen-Entwicklung eine Freude machen: NetBeans unterstützt nun JavaFX in der Version 2.0 (das JavaFX SDK wird allerdings nicht mit ausgeliefert, sondern muss erst noch von javafx.com heruntergeladen werden). Weitere Neuerungen in diesem Release sind unter anderem CSS3-Unterstützung und „grafisches Debuggen" in Swing und JavaFX: Der Nutzer erstellt während einer Debug-Sitzung Screenshots der Oberfläche und kann dann in der IDE diese Screenshots analysieren und die Eigenschaften der einzelnen Komponenten einsehen. Aber auch für Nicht-UI-Entwickler gibt es einiges, zum Beispiel verbesserte Werkzeuge für Refactorings (Batch Refactoring) sowie verbesserte Unterstützung für Maven und für verschiedene Versionskontroll-Systeme (Git-Support ist jetzt direkt mit dabei). Die Integration mit Weblogic 12c und GlassFish Open Source Edition 3.1.1 ist getestet worden, daneben werden auch GlassFish Enterprise Server 2.1.1, Tomcat 7.0.22 und JBoss 6.0 als „known to run" aufgeführt.

Andreas Badelt
andreas.badelt@doag.org

Andreas Badelt ist Technology Architect bei Infosys Limited. Daneben organisiert er seit 2001 ehrenamtlich die Special Interest Group (SIG) Development sowie die SIG Java der DOAG Deutsche ORACLE-Anwendergruppe e.V.

UPDATE

An dieser Stelle erhalten Sie in jeder Ausgabe ein Update über das Geschehen in der Java-Comunity

Pressemeldung von Oracle zum WebLogic Server 12c

Oracle kündigt die neueste Version von WebLogic Server, 12c, dem führenden Anwendungsserver für konventionelle Systeme, Engineered Systeme und Cloud-Umgebungen an. Als das Herzstück der Oracle Cloud Application Foundation und als eines der Kernelemente der Oracle-Fusion-Middleware-Produktfamilie bietet er in der neuen Version innovative Funktionen für die Entwicklung, die Nutzung und den Betrieb von Java EE-Applikationen.

Dank wesentlicher Erweiterungen können laut Oracle Kunden und Partner ihre Total Cost of Ownership reduzieren und ihre bestehende Applikationsinfrastruktur besser ausnutzen. Gleichzeitig können sie den Entwicklungszyklus ihrer Applikationen und deren Markteinführung beschleunigen. Weitere Features sind:

- Oracle WebLogic Server 12c ist für die komplette Spezifikation der Java EE 6 Plattform zertifiziert. Dies steigert die Produktivität bei der Entwicklung von Standard-basierten, modernen APIs – darunter Servlet 3.0, JAX-RS 1.1, Java Server Faces 2.1, EJB 3.1, Context und Dependency Injection for Java sowie viele andere.
- Entwickler, die Oracle WebLogic Server einsetzen, können die Features der Java Platform Standard Edition (Java SE) 7 noch besser dazu nutzen, Fehler im Code zu vermeiden und Code zu schreiben, der sich einfacher warten lässt.
- Oracle WebLogic Server bietet Out-of-the-Box Unterstützung für das Dependency Management und einen einheitlichen Entwicklungsprozess über ein aktualisiertes Oracle WebLogic Server Plug-in für Apache Maven.
- Die neue Fusion-Middleware-Komponente, Oracle Traffic Director (OTD), lässt sich direkt mit Oracle WebLogic Server 12c integrieren. Damit erhalten Kunden leistungsstarke und hochverfügbare Traffic-Routing-Möglichkeiten sowie dynamisch konfigurierbares Caching, Load-Balancing und Proxy Support für HTTP-basierte Applikationen.
- Mittels grafischer Werkzeuge und offener PaaS Web Service APIs sorgt Oracle Virtual Assembly Builder für eine einfachere Konfiguration. Er ermöglicht zudem die einfache Paketierung von Multi-Tier Enterprise-Applikationen für Virtualisierungsumgebungen unter Oracle VM.

Dank neuer Disaster-Recovery-Möglichkeiten können Kunden Daten entweder in einem File Store oder in einer Datenbank speichern. Es besteht auch die Möglichkeit, Transaction Logs in einer Datenbank zu speichern. Die Nutzung von Datenbank-integrierten konsistenten Replikationstechnologien mit Oracle Active Data Guard und Oracle GoldenGate wird für alle dynamischen Anwendungsdaten einschließlich Online Logs, Java Message Service (JMS) Logs sowie Transaction Logs ermöglicht. Auch die Applikationssicherheit wurde durch die Nutzung von Transport Layer Security (TLS) 1.2, dem Nachfolger von SSL, verbessert. Ein nahtloses Upgrade von Oracle WebLogic Server 11*g* ist gewährleistet.

Im Vergleich zu traditionellen Web-basierten Architekturen ermöglicht Oracle Traffic Director eine drei- bis vierfache Performance-Steigerung für Applikationen, die auf Oracle WebLogic Server und Oracle Fusion Middleware laufen. Der Oracle Traffic Director wird auf Oracle Exalogic Elastic Cloud bereitgestellt. Für maximale Kompression und SSL Performance kommen Advanced Encryption Set sowie Integrated Performance Primitives von Intel zum Einsatz.

Oracle WebLogic Server wurde für den Einsatz auf Oracle Exalogic Elastic Cloud optimiert, dem weltweit ersten und einzigen Engineered System für Cloud Computing. Oracle Exalogic Elastic Cloud wurde von Oracle getestet und so angepasst, dass es die bestmögliche Grundlage für Java Applikationen, Oracle Applikationen und andere Enterprise-Applikationen mit hervorragender Leistung bietet.

Oracle WebLogic Server ist darüber hinaus eine Schlüssel-Komponente des neuen Oracle Java Cloud Service. Diese Enterprise-Plattform für die Entwicklung, Bereitstellung und Verwaltung geschäftskritischer Java EE Applikationen unterstützt die Entwicklung und die Implementierung auf mehreren Java-basierten, integrierten Entwicklungsumgebungen, darunter Oracle JDeveloper, NetBeans IDE und Eclipse.

Cameron Purdy, Vice President Development bei Oracle, erläutert dazu: „Mit der neuen Version des führenden Anwendungsservers können unsere Kunden jetzt noch mehr aus ihrer bestehenden Infrastruktur herausholen, die Entwicklung und Verwaltung von Applikationen wesentlich verbessern und ihre Applikationen schneller zur Marktreife bringen. Der Grund dafür sind Innovationen, welche die Effizienz der Entwickler erhöhen. Darüber hinaus sind Kunden mit Oracle WebLogic Server 12c jetzt noch besser für Cloud Computing aufgestellt. Sie können ihre bestehende Infrastruktur optimal nutzen, um private oder öffentliche Cloud-Architekturen zu entwickeln. Oracle WebLogic Server und Oracle Exalogic Elastic Cloud bilden die ideale Basis für Cloud-Anwendungen mit maximaler Leistung, Skalierbarkeit und Zuverlässigkeit."

Die jüngsten Entwicklungen im Rechtsstreit um Android

Andreas Badelt, Leiter SIG Java, DOAG Deutsche ORACLE-Anwendergruppe e.V.

Im Laufe der letzten Monate hat sich viel getan im Rechtsstreit zwischen Oracle und Google um das Betriebssystem Android – zeitweise gab es täglich Neuigkeiten. Daher haben wir dieses Thema aus dem Java-Tagebuch herausgetrennt, um es hier in gebündelter Form zu präsentieren.

November 2011: Google hat Einspruch beim Berufungsgericht für Patentfragen eingelegt. Die „Lindholm-E-Mails", die (wie in der letzten Ausgabe berichtet) auf eine bewusste Rechtsverletzung durch Google schließen lassen, sollen zum Prozess nicht zugelassen werden, da sie an den Rechtsbeistand gerichtet gewesen seien und nicht als Beweismittel dienen dürfen. Richter Alsup hatte dies mehrfach abgelehnt. Die Wahrscheinlichkeit, dass das Berufungsgericht tatsächlich in ein laufendes Verfahren eingreift, ohne dass dessen Richter gravierende Fehler unterlaufen sind, scheint auch eher gering zu sein.

Aber es gibt ebenso ermutigende Signale für Google: Besagte E-Mails werden vermutlich erst in der dritten Phase des Verfahrens behandelt. Diese findet nur dann überhaupt statt, wenn Oracle das Gericht tatsächlich von einer Verletzung von Urheberrechten (Phase eins) oder Patenten (Phase zwei) durch Google überzeugen kann – also wenn es nur noch darum geht, ob diese auch vorsätzlich stattgefunden haben, sowie um die Festlegung der Schadenshöhe (die durch den Vorsatz beeinflusst wird).

Die Aufteilung des Verfahrens ist auch weiterhin Teil der Diskussionen mit dem Richter. Oracle möchte, dass diese zumindest direkt aneinander anschließen und die Beweisaufnahme für alle gemeinsam stattfindet. Damit sollen laut Oracle Mehrfachvorladungen von Zeugen vermieden und komplexe, übergreifende Zusammenhänge besser dargestellt werden können. Bislang plant Richter Alsup mit längeren Pausen zwischen jeweils in sich abgeschlossenen Phasen. Google auf der anderen Seite möchte Phase zwei (Patentverletzungen) komplett ausklammern, solange die Gültigkeit der Patente nicht abschließend geklärt ist.

Ebenfalls eine wichtige Rolle in den Verhandlungen spielt natürlich der generelle Starttermin des Verfahrens. Nachdem dieses aufgrund einer Terminkollision mit einem Strafverfahren auf 2012 verschoben werden musste, drängt Oracle auf einen Start noch im Januar, während Google auf Zeit spielt (dies aber natürlich nicht öffentlich zugeben kann). Die Anrufung des Berufungsgerichts und die Anträge zur zweiten Phase sollen vermutlich auch dabei helfen.

Der Nikolaus bringt diesmal keine Geschenke für Oracle: Richter Alsup hat am 6. Dezember 2011 einem Antrag von Google teilweise stattgegeben und den von Iain Cockburn verfassten Schadensbericht um mehrere wesentliche Punkte reduziert. Beide Parteien können aber innerhalb von vierzehn Tagen Gründe vorlegen, um diese Entscheidung noch zu ändern.

Kurz vor Weihnachten bestätigt das US-Patentamt dann die Ungültigkeit eines zweiten der sechs noch für das Verfahren relevanten Patente (noch zwei weitere wurden auch schon als „ungültig" erklärt, die Bestätigung steht für diese jedoch noch aus). Ein weiterer Punkt für Google – allerdings ist auch dies keine endgültige Entscheidung, sondern kann und wird sicherlich von Oracle angefochten werden.

Am 4. Januar 2012 bestätigt Richter Alsup die Ansetzung des Verfahrens: Es soll wie geplant in drei Phasen stattfinden und am oder nach dem 19. März 2012 beginnen. Über die Details wird bis dahin aber wohl weiter gestritten.

Hinweis: Wer sich für mehr Details zur Android-Klage und ähnliche Themen interessiert, dem sei das Blog von Florian Müller ans Herz gelegt, das nicht nur für den vorliegenden Artikel, sondern auch für Zeitungen und Magazine in aller Welt eine unschätzbare Informationsquelle ist: http://fosspatents.blogspot.com

Andreas Badelt
andreas.badelt@doag.org

Andreas Badelt ist Technology Architect bei Infosys Limited. Daneben organisiert er seit 2001 ehrenamtlich die Special Interest Group (SIG) Development sowie die SIG Java der DOAG Deutsche ORACLE-Anwendergruppe e.V.

Aus Alt mach Neu: Do's and Don'ts bei der Forms2Java-Migration

Björn Christoph Fischer und Oliver Zandner, Triestram & Partner GmbH

Oracle Forms ist bekannt und geschätzt für seine außerordentliche Produktivität in der Entwicklung und seine Stabilität im Betrieb. Allerdings müssen sich Forms-Anwendungen im Business-Umfeld immer mehr an dem messen lassen, was Business-User aus ihrer Freizeit und dem Internet kennen: Rich-Client-Applikationen, die sehr ergonomisch zu bedienen sind, die der Benutzer an seine Bedürfnisse anpassen kann, die die Daten interaktiv visualisieren und die sogar via Internet von überall aus aufgerufen werden können.

Rich-Client-Anwendungen prägen die Erwartungshaltung von Business-Benutzern und setzen neue Standards im Bereich der Usability. Daraus erwächst ein spürbarer Druck auf Software-Produkte auf Forms-Basis, diese Erwartungshaltung ebenfalls zu erfüllen. Die Triestram & Partner GmbH (t&p) entwickelt eine solches Standard-Software-Produkt namens „lisa.lims" für den Einsatz im Labor. Dabei handelt es sich um eine typische Online-Transaction-Processing-Anwendung (OLTP), bei der es darum geht, im Mehrbenutzer-Betrieb Daten in Dialogen konsistent zu erfassen und automatisch von externen Quellen wie zum Beispiel Labor-Geräten über Schnittstellen zu übernehmen, die Daten zu bearbeiten, zu validieren und auszuwerten sowie in Berichten zu veröffentlichen. Bis zur Version 9 basierte die Software auf der Oracle-Datenbank mit rund vierhundert PL/SQL-Packages, etwa 150 Oracle Forms und circa 250 Oracle Reports. Für die Version 10 wurden – unter Beibehaltung der Oracle-Datenbank – die Dialoge und die Berichte auf Java-Open-Source-Technologien umgestellt, nämlich:

- Die Eclipse Rich Client Platform (RCP) für die Desktop-Version der Dialoge
- Die Eclipse Rich Ajax Platform (RAP) für die Web-Version der Dialoge
- Die Business Intelligence and Reporting Tools (BIRT) für die Berichte

Technische Schlüssel-Anforderungen an die migrierte Anwendung

Der eigentlichen Migration ging eine Technologie-Studie voran, bei der die strategischen und technischen Anforderungen an die migrierte Software erhoben und bewertet wurden. Die migrierte Anwendung soll Folgendes erfüllen:

1. Die vorhandene Business-Logik in PL/SQL in der Datenbank wiederverwenden, um die Investition zu schützen, die bislang dort hineingeflossen ist
2. Eine individualisierbare Benutzeroberfläche haben
3. Auf Java als Basis-Plattform setzen
4. Sowohl als Desktop- als auch als Web-Anwendung verfügbar sein
5. In ihrer Web-Variante mindestens mit dem Microsoft Internet Explorer und Mozilla Firefox kompatibel sein
6. Mit möglichst vielen Applikations-Servern kompatibel sein

Zunächst wurde versucht, diese Anforderung auf Basis der Java-Enterprise-Architektur (JEE) mit einem JEE-konformen Application-Server umzusetzen. Dabei ist das JEE-Paradigma mittelschichtzentriert: Daten werden aus der Datenbank in den Application-Server auf der Mittelschicht geladen und die Anwendung verändert die Daten ausschließlich über den Application-Server. Die Konsistenz der Daten wird vom Application-Server sichergestellt, Transaktionen werden üblicherweise ebenfalls von diesem verwaltet. Die Pilot-Migration der vorhandenen Forms-Anwendung in eine JEE-Architektur scheiterte unter anderem daran, dass es im Umfeld von lisa.lims außer dem Anwendungs-UI noch weitere Systeme – wie zum Beispiel Laborgeräte – gibt. Diese Geräte schreiben ihre Messdaten mittels PL/SQL-Aufrufen direkt in die Datenbank, also ohne den Weg über den Application-Server zu gehen. Die Kombination aus JEE-konformem Application-Server plus „Satelliten" mit direktem Schreibzugriff auf die Datenbank stellte sich als inkompatibel heraus, da beides zusammen die Konsistenz der Daten in der Datenbank gefährdet.

Welche Relevanz hat das für die Migration des User Interface (UI) von Forms nach Java? Forms ist ein UI-Framework, das dem Entwickler bestimmte UI-Standard-Funktionalitäten „frei Haus" zur Verfügung stellt, zum Beispiel:

- Die Bedienbarkeit der Anwendung per Funktionstasten
- Die Synchronisation mit der Datenbank, wenn der Benutzer Datensätze in die Anzeige einfügt, sie ändert oder löscht
- Die Validierung eingegebener Werte (etwa per „WHEN-VALIDATE"-Trigger)
- Die Synchronisation von Master- und zugehörigen Detail-Datensätzen in der Anzeige
- Die Möglichkeit, an beliebigen Stellen der Anwendung SQL-Abfragen abzusetzen
- Die Möglichkeit, Datensätze in die Anzeige „lazy" nachzuladen, wenn der Benutzer anfängt, in der Ergebnis-Liste zu scrollen

Wenngleich der mittelschichtzentrierte Ansatz von JEE für Anwendungen, die „auf der grünen Wiese" entstehen, sicherlich vorteilhaft ist, ergab die Technologie-Studie jedoch, dass die OLTP-Anforderungen von lisa.lims nicht mit der JEE-Architektur umsetzbar waren. Insbesondere das pessimistische Locking und die Wiederver-

wendung der vorhandenen PL/SQL-Packages verhinderten den Einsatz von JEE für das Migrationsprojekt. Also mussten die beschriebenen Funktionalitäten mit anderen Technologien als JEE umgesetzt werden. Zum Zeitpunkt der Evaluation (2008) gab es außerhalb von JEE keine Java-Frameworks, die die beschriebenen datenbankorientierten UI-Anforderungen „von der Stange" erfüllt hätten. Der Artikel beschreibt, wie die Anforderungen mit einem eigenen Framework („rapid.Java") in Java außerhalb von JEE umgesetzt wurden. Der Fokus liegt hierbei auf der Implementierung der Client-Funktionalitäten, die das neu zu entwickelnde Framework im Rahmen seiner Model-View-Controller-Architektur bieten musste, um die gegebenen Anforderungen zu erfüllen.

Datensatz-Management im UI

Datensatz-Management bedeutet, dass der Controller den Zustand der Datensätze im UI kennen und auf diese verschiedenen Zustände reagieren muss – zum Beispiel: Wurde ein Datensatz vom Benutzer neu in die Anzeige eingefügt, muss der Controller diesen auch dem Model hinzufügen. Wird ein aus der Datenbank gelesener Datensatz in der Anzeige verändert, muss der Controller zunächst prüfen, ob der Datensatz nicht eventuell bereits durch einen anderen Benutzer verändert worden ist – falls ja, ist eine entsprechende Meldung auszugeben etc. Abbildung 1 gibt einen Überblick über die verschiedenen Status im Lebens-Zyklus eines Datensatzes.

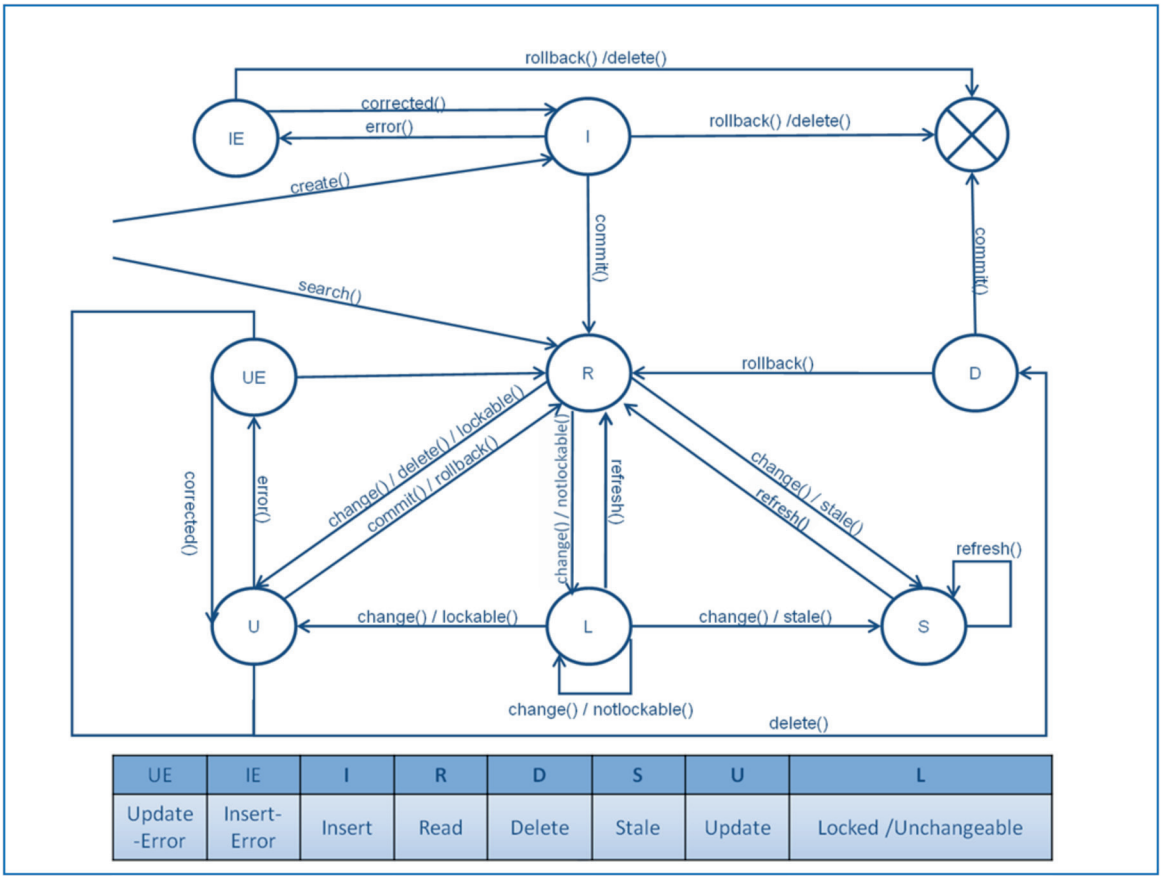
Das Management dieser verschiedenen Zustände ist einer der Automatismen, die Forms dem Entwickler bietet und der bei der Migration von Forms nach Java zu berücksichtigen ist. In unserem Migrations-Projekt wurde er als Teil der MVC-Architektur auf der Client-Seite implementiert und ist eine der Basis-Funktionalitäten des „rapid.Java"-Frameworks.

Master-Detail-Synchronisation

Die Master-Detail-Synchronisation betrifft die Frage, wie das UI Master-Detail-Relationen aus der Datenbank darstellt, wie etwa Aufträge und Auftragspositionen. Dabei sind zwei Use-Cases zu bedenken:

1. Wenn der Benutzer in der Ansicht von einem zum nächsten Master-Datensatz wechselt, müssen in der Detail-Ansicht automatisch die passenden Detail-Datensätze nachgeladen werden.
2. Was ist zu tun, wenn der Benutzer einen Detail-Datensatz ändert und anschließend in der Master-Ansicht weiterscrollt? Hier ist eine Warnung notwendig, die auf ungespeicherte Änderungen aufmerksam macht.

Diese Anforderungen wurden in rapid.Java wie folgt gelöst: Für jedes Anzeige-Element im UI (Tabelle oder Formular) existiert im Model eine Instanz einer speziellen Container-Klasse, die in rapid.Java als „EntityCollection" bezeichnet wird. Die Relation zwischen den Datensätzen aus der Datenbank wird durch eine in XML abgebildete Baumstruktur definiert, welche die existierenden EntityCollections hierarchisch gliedert. Dadurch kennt der Controller die Abhängigkeiten zwischen den Objekten im Model und weiß, welche Details jeweils nachzuladen sind, wenn in der View von einem zum nächsten Mas-



UE	IE	I	R	D	S	U	L
Update -Error	Insert-Error	Insert	Read	Delete	Stale	Update	Locked /Unchangeable

Abbildung 1: Die möglichen Zustände im Lebenszyklus eines Datensatzes

ter-Datensatz gescrollt wird. Der Controller implementiert ebenfalls das Tracking geänderter Detail-Datensätze und löst entsprechende Prüfungen und Warnhinweise aus, wenn im Master-Datensatz gescrollt wird.

Validierungen

Forms sorgt durch seine Validierungs-Trigger dafür, dass der Cursor im aktuellen Feld „festgehalten" wird, wenn der Benutzer dort eine inkonsistente Eingabe vorgenommen hat. Wie soll in solchen Situationen in der migrierten Java-Anwendung verfahren werden?

Die Erfahrung zeigt, dass sich das Validierungsverhalten von Forms mit der gewählten UI-Bibliothek nur schwer umsetzen lässt. Das liegt unter anderem daran, dass editierbare Tabellen-Ansichten, sogenannte „Data-Grids", an sich schon eine komplizierte Logik zum Aktivieren und Deaktivieren von Zell-Editoren abbilden müssen. In diese Logik einzugreifen und das Verlassen einer Zelle zu verhindern, hat sich als extrem schwierig herausgestellt. Außerdem wird in Eclipse RCP das Rendering der Steuerelemente vom darunterliegenden Betriebssystem vorgenommen. Damit ist die Steuerung des Fokus im Falle einer fehlgeschlagenen Validierung ebenfalls vom Betriebssystem abhängig, was bei der Realisierung des Festhaltens immer wieder zu Problemen führte. Deshalb lautet das Fazit in diesem Punkt, dass das Forms-Verhalten bei Validierungen auf keinen Fall in der migrierten Java-Anwendung imitiert werden sollte, sofern man nicht die absolute Kontrolle über das Verhalten der Steuerelemente der UI-Bibliothek hat. Mit dem Verzicht auf dieses Verhalten stellen sich aber auch die folgenden Fragen, die es zu beantworten galt:

- Wie bemerkt der Benutzer überhaupt, dass er einen Eingabe-Fehler gemacht hat?
- Wie soll mit mehreren Fehlern innerhalb eines Datensatzes umgegangen werden?
- Wie soll mit mehreren Fehlern an verschiedenen Datensätzen umgegangen werden?
- Wo werden die fehlerhaften Werte zwischengespeichert, bis der Benutzer sie korrigiert?

Im Migrations-Projekt wurden diese Probleme wie folgt im rapid.Java-Framework gelöst:

1. Der Benutzer kann mehrere Fehler bei der Eingabe machen, aber nur in einem Datensatz. Existieren Fehler, ist ausschließlich der fehlerhafte Datensatz noch editierbar.
2. Wenn der Benutzer den fehlerhaften Datensatz mit dem Cursor verlässt, wird ein modaler Dialog mit einer Fehlerliste angezeigt, sodass kein Fehler übersehen wird.
3. Solange die Fehler im aktuellen Datensatz existieren, werden neue Eingaben, wie Speichern, Einfügen, Löschen und Wechseln zu einem anderen Master-Datensatz, verhindert.

Paging oder kein Paging bei einer Forms-Ablösung

In Forms lässt sich per Parameter bestimmen, in welchen Paketgrößen Datensätze beim Scrollen nachgeladen werden, wenn die Größe der Treffermenge die Anzahl der anzeigbaren Datensätze übersteigt (Paging). Da dieses Verhalten Vor- und Nachteile besitzt, muss bei einer Migration nach Java entschieden werden, ob ein äquivalentes Paging im Framework umgesetzt wird oder alternativ auf Paging – und damit auch auf dessen Vorteile – verzichtet werden kann.

Im rapid.Java-Framework wird auf das Paging verzichtet und stattdessen die gesamte Treffermenge auf den Client geladen. Übersteigt die Treffer-Menge einen einstellbaren Schwellwert X, wird vor dem Ausführen der Datenbank-Abfrage vom Benutzer erfragt, ob die Treffermenge von der Größe X tatsächlich angezeigt werden soll. Bei Überschreiten eines systemweit einstellbaren Maximalwertes (zum Beispiel 50.000 Datensätze) wird die Ausführung der Abfrage komplett abgewiesen. Der Verzicht auf das Paging bietet folgende Vorteile:

- Die Sortierung auf dem Client erfordert keine erneute Datenbank-Abfrage, wenn die gesamte zu sortierende Treffermenge bereits in das Model geladen wurde.
- Ebenso sind Gruppen-Funktionen auf dem Client ohne erneute Abfrage zu berechnen.
- Das Scrollen in großen Treffermengen ist performanter, weil die Datensätze nicht nachgeladen werden müssen.

In einem Java-Client SQL-Abfragen absetzen

Forms bietet die Möglichkeit, an beliebigen Stellen im Code SQL-Abfragen an die Datenbank abzusetzen. In Java sind dafür folgende Möglichkeiten denkbar:

1. *Die Java Persistence Query Language (JPQL)*
   JPQL ermöglicht es, über einen Objekt-relationalen Mapper Abfragen an die Datenbank abzusetzen, die mit Objekten (Entity Beans beziehungsweise Persistent Entities) und deren Attributen arbeiten statt mit Tabellen und Spalten. Grundlage hierfür ist die Definition eines Mappings von Datenbank-Tabellen auf Entity-Klassen in Java. Die Syntax von JPQL ist an SQL angelehnt und ermöglicht es, von der eigentlichen Abfrage-Sprache der Datenhaltungs-Schicht zu abstrahieren.
2. *Natives SQL*
   Die native Abfrage-Sprache der jeweiligen Datenhaltungs-Schicht, in der Regel also ein SQL-Dialekt. Die entsprechenden SQL-Statements werden als Strings in die Anwendung integriert und via JDBC oder über JPA (Java Persistence API) als „Native Queries" an die Datenbank gesendet.
3. *Eigene Java-API*
   In einer Drei-Tier-Architektur kann auf Server-Seite eine selbstdefinierte API implementiert werden, welche die Ausführung von Abfragen kapselt. So kann die Komplexität von JPQL beziehungsweise SQL verborgen und eine einfache Möglichkeit geschaffen werden, aus dem Java-Client heraus Abfragen abzusetzen.

Aufgrund der gewählten Drei-Schicht-Architektur (Datenbank, Server-Komponente und Client) war die direkte Verwendung des OR-Mappers oder JDBC auf dem Client nicht möglich. Stattdessen bringt rapid.Java eine eigene API mit („rapid.Java Persistence API"). Diese stellt vereinfachte Abfrage- und DML-Mechanismen bereit, wobei für besondere Situationen auch die Ausführung von JPQL oder SQL über den Server ermöglicht wird. Die API enthält:

- Eine zentrale Server-Komponente für DML-Operationen (INSERT, UPDATE, DELETE)
- Eine Methode namens „find()“, die Abfragen in der Datenbank auf Basis des Primär-Schlüssels ermöglicht
- Eine Familie überladener Methoden namens „findByCriteria()“, die Abfragen in der Datenbank auf Basis anderer als der Primär-Schlüssel-Felder ermöglichen und dabei weniger komplex, aber auch weniger flexibel sind als JPQL- oder SQL-Abfragen
- Verschiedene Methoden zur Ausführung von JPQL- oder Native Queries, die für sehr komplizierte oder Performance-kritische Abfragen verwendet werden können

Auf diese Weise können Standardfälle schnell mit dem vereinfachten API abgebildet werden, ohne dass die Flexibilität verloren geht, die für komplexere Abfragen notwendig ist.

Häufige UI-Anforderungen, die Forms nicht bietet

Der Einsatz einer neuen Technologie ist nicht ausschließlich ein Ersatz für Forms, sondern bietet darüber hinaus Funktionalitäten, die in Forms bislang nicht verfügbar und umsetzbar waren. Im Folgenden wird beschrieben, wie diese zusätzlichen UI-Anforderungen im Framework realisiert wurden.

In Forms bestehen Tabellen aus einzelnen Formular-Steuerelementen, die vom Forms-Developer in Form einer Matrix angeordnet werden. Das bringt die folgenden Nachteile mit sich:

- Die Inhalte der Spalten sind nicht automatisch per Mausklick auf den Spaltenkopf sortierbar
- Die Spaltenbreite kann zur Laufzeit nicht durch den Benutzer verändert werden
- Die Reihenfolge der Spalten kann nicht zur Laufzeit durch den Nutzer geändert werden

In rapid.Java basiert das UI auf dem Standard Widget Toolkit (SWT) von Eclipse und der JFace-Bibliothek, die auf SWT aufsetzt und zusätzliche UI-Komponenten zur Verfügung stellt. SWT enthält ein UI-Element für die Darstellung von Data Grids, das sogenannte „Table Widget“. Das Anpassen der Reihenfolge und der Breiten der Spalten ist bereits ein Feature des SWT.

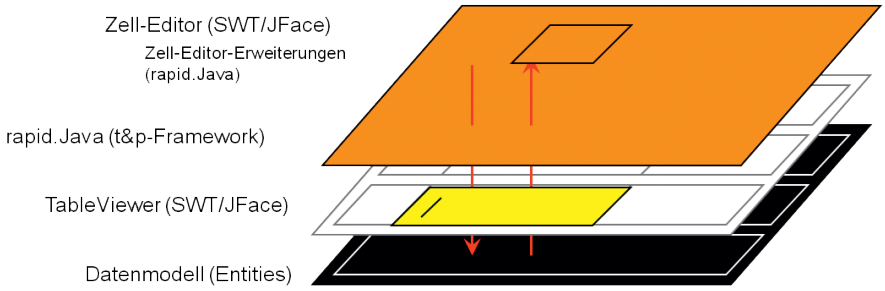


Abbildung 2: Editierbare Tabellen im rapid.Java-Framework durch Erweiterung von SWT

Die JFace-Bibliothek bietet zusätzlich eine Möglichkeit, Tabellen automatisch per Klick auf den Spalten-Header zu sortieren. Diese Sortier-Lösung war jedoch für das rapid.Java-Framework nicht ausreichend, da sie sich lediglich auf die Ansicht der Daten, nicht aber auf die Reihenfolge der Entitäten im Model bezieht. Das ist immer dann problematisch, wenn eine Aktion an einer bestimmten Stelle in der Folge der Datensätze erfolgen muss (zum Beispiel Einfügen eines neuen Satzes zwischen zwei vorhandenen). Eine Differenz der Sortierungen im UI und im Model würde bei solchen Aktionen zu falschen Ergebnissen führen. Deshalb wurde die Sortierbarkeit von Tabellen in rapid.Java so implementiert, dass die Sortierung direkt im Model abgebildet wird. Dadurch ist anschließend auch die Anzeige neu sortiert und Anzeige und Model bleiben weiterhin synchron.

Das Editieren von Tabellen im UI basiert in rapid.Java auf dem „Editing-Support“ von JFace. Dieser wurde um folgende Funktionalitäten erweitert (siehe Abbildung 2):

- *Sperren der editierten Datensätze in der Datenbank*
  Sobald ein Benutzer Datensätze in der Table editiert, werden diese von Java/RCP nicht standardmäßig in der Datenbank gesperrt. Das Locking wurde deshalb in rapid.Java implementiert.
- *Komfortable Editierungs-Möglichkeiten*
  Die Zell-Editoren der JFace-Bibliothek wurden so erweitert, dass dem Benutzer komfortable Möglichkeiten zum Editieren der Daten zur Verfügung gestellt werden können. Hierzu gehören neben dem einfachen Text-Zell-Editor auch Combo- und Checkboxen sowie ein Dialog-Zell-Editor, um Werte komfortabel auszuwählen („List of values“, „Date Picker“-Dialog oder komplexe, selbstdefinierte Auswahl-Dialoge).

UI-Konfiguration

Die Konfigurierbarkeit des UI ist eine typische Anforderung aus der Entwicklung einer Standard-Software. Sie ermöglicht es, das UI an Kunden-Spezifika anzupassen, ohne dafür den Quellcode der Anwendung verändern zu müssen. Gleichzeitig ist eine sehr weitgehende Konfigurierbarkeit des UI aus einem Datenbank-Repository in vielen Java-Anwendungen eher unüblich.

In rapid.Java ist insbesondere die Darstellung und Funktionalität von Feldern, Tabellen- und Formularansichten konfigurierbar. Das Erscheinungsbild und das Verhalten beruhen dabei auf Konfigurations-Tabellen in der Datenbank:

- *Konfiguration von Tabellen im UI*
  Hierbei wird in der Datenbank angegeben, auf welcher Datenbank-Tabelle die Tabelle im UI beruht und welche ihrer Spalten wie dargestellt werden sollen. Im Code der Anwendung genügt dann der Aufruf einer speziellen Framework-Methode („createTable()“), den Rest übernimmt das rapid.Java-Framework.
- *Konfiguration von Formularen im UI*
  Bei Formularen werden deren Layout und die Eigenschaften der Felder in der Datenbank abgelegt. Für jedes anzuzeigende Feld ist im Anwendungscode

lediglich ein Aufruf der „createField()"-Methode nötig; die Konfiguration des Feldes wird von rapid.Java vorgenommen.

Die folgenden Eigenschaften von Feldern in Tabellen oder Formularen können per Konfiguration in der Datenbank festgelegt werden:

- Welches Attribut von welcher Entity-Klasse soll angezeigt/bearbeitet werden?
- Von welchem Typ soll das Feld sein (Text, Combo-Box, Check-Box) und welchen Daten-Typ soll es enthalten (String, Date, Number, Boolean)?
- Welche visuellen Attribute soll es haben (Hintergrund- und Vordergrund-Farbe, Schriftart)?
- Welche Beschränkungen oder Funktionalitäten gibt es bei der Eingabe (maximale Eingabe-Länge, erlaubte Zeichen, „List of Values", Pflicht-Feld, Zahlen- oder Datumsformat usw.)
- Welche Validierungsregeln sollen beim Verlassen des Feldes angewandt werden („Regular Expression" oder selbstdefinierte Validierung per SQL-Abfrage)?

Internationalisierung

Die Internationalisierung einer Anwendung umfasst neben den Übersetzungen sämtlicher Texte im UI (Labels, Meldungstexte etc.) auch die landesspezifische Formatierung von Zahlen und Datumswerten. In Java-Anwendungen werden für gewöhnlich die im Betriebssystem konfigurierten Landes- und Spracheinstellungen („Locale") verwendet, um Texte aus Properties-Dateien zu übersetzen, die mit der Anwendung ausgeliefert werden. Da eine Anforderung an das rapid.Java darin bestand, Sprach- und Länder-Kennzeichen aus den Benutzer-Einstellungen in der Datenbank zu verwenden sowie auch die Übersetzungen ausschließlich zentral in der Datenbank statt dezentral im Dateisystem vorzunehmen, wurde ein eigener Ansatz gewählt.

Basierend auf dem Sprach- und Länder-Kennzeichen in der Datenbank erstellt das Framework beim Login ein passendes „Locale", das im Verlauf der Sitzung zur Erstellung von Datums- und Zahlenformaten auf dem Client verwendet wird. Zusätzlich berücksichtigt der PL/SQL-Teil des Konfigurations-Frameworks die Kennzeichen und gibt Texte für Labels, Meldungen etc. bereits in der korrekten Benutzersprache zurück. So muss der Übersetzungsvorgang nicht mehr separat auf dem Client durchgeführt werden, sondern wird automatisch von rapid.Java vorgenommen.

Kernanforderungen

In Forms werden Datensätze für die Dauer ihrer Bearbeitung automatisch in der Datenbank gesperrt („Pessimistic Locking"). Auf diese Weise stellt Forms sicher, dass die Daten in der Datenbank im Mehrbenutzer-Betrieb konsistent bleiben. Die Übertragung dieses Verhaltens in eine JEE-konforme Architektur scheiterte im Migrations-Projekt unter anderem an zwei Automatismen der Application-Server, nämlich am Caching der Daten aus der Datenbank auf der Mittelschicht und am „Connection-Pooling". Dieser Mechanismus bedeutet, dass beim Start des Application-Servers eine gewisse Anzahl von Datenbank-Sitzungen gestartet wird. Führt ein Benutzer in der Anwendung eine datenbankbezogene Aktion aus, wird ihm dazu eine der Verbindungen aus dem Pool zugewiesen, die nach Beendigung der Aktion wieder in den Pool zurückgegeben wird. Eine Konsequenz ist, dass alle Benutzer der Anwendung mit demselben Datenbank-Benutzer angemeldet werden, was bei vorhandenen PL/SQL-Packages Probleme bereiten kann. Diese Mechanismen verhinderten in unserem Fall auch die Integration von PL/SQL in der Datenbank in eine JEE-Architektur. Daher wurde im rapid.Java-Framework eine Infrastruktur für das „Pessimistic Locking" und die Integration von PL/SQL in der Datenbank implementiert.

Fazit

Die Migration einer Individual-Software von Oracle Forms nach Java ist anspruchsvoller und komplexer als zunächst angenommen. Das liegt vor allem daran, dass Oracle Forms die Entwicklung von OLTP-Anwendungen durch zahlreiche Automatismen und eine entsprechende Infrastruktur stark vereinfacht. Für diese Automatismen und diese Infrastruktur müssen in der Java-Welt Äquivalente gefunden werden. Wo es diese in der Java-Welt nicht gibt, müssen sie selbst implementiert werden.

Zum Zeitpunkt der Technologie-Evaluation gab es keine Open-Source-Frameworks in Java, die die oben genannten, typischen OLTP-Features von Oracle Forms von der Stange geboten hätten. Sicherlich: Es gab zu diesem Zeitpunkt schon kommerzielle Java-Frameworks wie Oracle ADF. Diese basieren jedoch auf einer wesentlich schwergewichtigeren Infrastruktur als der nun geschaffene Ansatz von rapid.Java.

Aufgrund dieser Umstände hat sich t&p dazu entschlossen, ein eigenes, leichtgewichtiges Framework für die Forms2Java-Migration zu entwickeln. Dieses Paper hat im Sinne von „Do's and Don'ts" gezeigt, welche Punkte bei der Framework-Entwicklung zu berücksichtigen sind, welche Wege man gehen sollte und welche Pfade besser zu vermeiden sind, um an das gewünschte Ziel zu gelangen.

Björn Christoph Fischer
b.fischer@t-p.com
Oliver Zandner
o.zandner@t-p.com

Oliver Zandner ist bei Triestram & Partner zuständig für Marketing und Vertrieb von Individual-Lösungen und -Projekten auf Basis der Plattformen Java, Oracle und .Net. Ein aktueller Schwerpunkt ist das Marketing für das „rapid.Java development framework".

Björn Christoph Fischer ist Consultant bei Triestram & Partner (t&p). Er ist einer der Architekten des „rapid.Java development framework" von t&p – ein Java-Framework für das Rapid Application Development und die schrittweise Migration von Oracle-Forms-Anwendungen nach Java. Schwerpunktmäßig befasst er sich dabei mit Eclipse-Technologien.

„IBM ist in vielen Standardisierungsgremien federführend …“

Für Java aktuell ist Chefredakteur Wolfgang Taschner im Gespräch mit John Duimovich von IBM Canada. Er ist Distinguished Engineer in der IBM Software Group mit Schwerpunkt „Java Virtual Machines & Embedded Java“. Außerdem hält er mehrere Patente im Bereich „Resource Management und Synchronisierung“.

Welchen Stellenwert hat Java bei IBM?
Java wird in Hunderten von IBM-Software-Produkten einschließlich WebSphere Application Server als Basis-Technologie verwendet und ist somit für IBM und unsere Kunden von zentraler Bedeutung. Außerdem bildet Java die Grundlage für unsere Tools-Plattform, die Eclipse als Basis zur Erstellung der Rational-Jazz-Kollaborations-Plattform für die Software-Entwicklung nutzt. Java ist darüber hinaus das Herzstück der hochleistungsfähigen und skalierbaren Produkte für unsere Mainframes mit Java-Umgebung (System z) und Power-Prozessorfamilie, die zu den Schlüsselkomponenten der Smarter-Planet-Initiative von IBM gehören. Eine Vielzahl von IBM-Produkten ist von der skalierbaren und portierbaren Infrastruktur der Java-Plattform abhängig, um die Flut an neuen Daten nutzen zu können, die von Sensoren erstellt werden, die heutzutage in allen Gebrauchsgegenständen von Autos und Waschmaschinen bis hin zu Stromnetzen sowie von RFID-Tags bis Twitter-Konversationen zu finden sind. Letztendlich sind es unsere Kunden, die von den IBM-Investitionen in Java-Technologien profitieren, und dieser Erfolg wird sich auch in der Zukunft fortsetzen.

Welche Ziele hat IBM mit Java?
IBM hat sich zum Ziel gesetzt, dafür zu sorgen, dass Java weiterhin eine bedeutende und innovative Anwendungs-Plattform bleibt. Dadurch wird gewährleistet, dass das Java-Ökosystem auch in Zukunft für Entwickler und Investitionen attraktiv bleibt und neue Funktionalität in eine Java-Plattform integriert wird, die offen und vernetzt sowie fähig ist, neue Lösungen zu tragen, auf die sich unsere Kunden verlassen können.

Wie beurteilen Sie die Lage im Enterprise-Java-Markt?
Der Enterprise-Java-Markt ist nach wie vor dynamisch und fördert Innovationen mit neuen Funktionen für Unternehmensanwendungen, die neu entstehende geschäftliche Herausforderungen adressieren. Die Java-Community trägt dazu bei, dass diese Innovationen allen unseren Kunden in standardisierten Lösungen zur Verfügung stehen. IBM ist in vielen Standardisierungsgremien federführend und bringt diese Führungsposition in die Enterprise-Java-Lösungen ein, die wir unseren Kunden anbieten.

Wie ist das Verhältnis zwischen IBM und Oracle?
IBM und Oracle verfolgen in Bezug auf die Java-Plattform viele gemeinsame Ziele und aus dieser gemeinsamen Ausrichtung ist eine produktive Beziehung entstanden. IBM unterhält schon seit langer Zeit eine enge Arbeitsbeziehung zu Oracle und anderen Hauptakteuren in der Java-Community. Wir werden auch künftig effektiv zusammenarbeiten. Die Vorteile dieser Ausrichtung sind eindeutig: Erst kürzlich haben wir durch unsere gemeinsame Arbeit am OpenJDK erreicht, dass IBM in der Lage war, die neuesten Java 7 SDKs schneller als je zuvor auf den Markt zu bringen.

Was erwarten Sie von Oracle in Bezug auf Java?
Wir erwarten von Oracle, dass sie die Java-Plattform und die Sprachen-Standards voranbringen. Wir sind fest davon überzeugt, dass leistungsfähige Führung und verstärkte Investitionen in den Java-Standard die Voraussetzung für eine kraftvolle und vitale Java-Community und ihr Ökosystem sind. Auf dieser Basis kann IBM die Beziehung zu Oracle ausbauen und die Innovationen voranbringen, die unsere Kunden für die Plattform fordern.

Welche Chancen sehen Sie für Android in den kommenden Jahren?
Wir sind der Überzeugung, dass in absehbarer Zukunft mehrere mobile Plattformen entstehen werden. IBM beabsichtigt, Produkte zur Ausführung auf diesen Plattformen bereitzustellen und unseren Kunden die geforderte Funktionalität zu bieten.

Was erwartet IBM in Zukunft von OpenJDK?
IBM geht davon aus, dass im Rahmen der Governance- und Entwicklungsprozesse die Offenheit und Transparenz des OpenJDK weiter ausgebaut werden. Für jedes Open-Source-Projekt, das als Idee eines einzelnen Unternehmens startete, besteht die permanente Herausforderung darin, über die ursprünglichen Ziele hinauszuwachsen und zu einem mehr auf Offenheit und auf Zusammenarbeit beruhenden Entwicklungsmodell zu finden. In dem Grade, in dem dies dem OpenJDK gelingt, wird sich eine noch größere Teilnehmer-Gemeinschaft an der Weiterentwicklung und Innovation der zentralen Java-Runtime beteiligen.

Wie interpretieren Sie die Entscheidung von Oracle, OpenJDK als Referenz-Implementierung für Java festzulegen?
Der Schritt von einem proprietären JDK hin zu einer Referenz-Implementierung, die vollständig auf dem OpenJDK-Projekt basiert, ist als positive Maßnahme für das Java-Ökosystem und unsere Kunden zu sehen. Damit ist ganz klar, welche Komponenten zur Standard-Java-Platform-Run-

time (RI) gehören und welche proprietäre Erweiterungen sind, die mit kommerziellen Java-Versionen geliefert werden.

Was erwarten Sie von der Community in Bezug auf Java?

Der unvergleichliche Erfolg von Java beruht weitestgehend auf seinem Erbe als Branchenbewegung gegenüber anderen proprietären Plattformen, die von einzelnen Herstellern kontrolliert werden. Die Mehrzahl der Innovationen im Java-Bereich entsteht heute durch Anstrengungen innerhalb der Community. Neue Sprachen wie Scala und X10, Entwicklungsplattformen für große Datenmengen (Big Data) wie Apache Hadoop, Werkzeuge und Runtime-Innovationen von Eclipse und vieles mehr prägen die Weiterentwicklung von Java. Wir erwarten, dass branchenübergreifende Zusammenarbeit, Entwicklung und Innovation das kontinuierliche Wachstum von Java und dessen Erfolg bis weit in die Zukunft sichern werden.

In welche Richtung sollte Java sich entwickeln?

Es gibt eine Reihe maßgeblicher Faktoren, die wir für die Weiterentwicklung von Java für wichtig halten: Der erste besteht darin, Entwicklern die Erstellung von Anwendungen zu ermöglichen und deren geschäftsbezogene Probleme einfach und schnell zu lösen. Innovationen, die sich an dieser Forderung orientieren – wie Einfachheit der Plattform und bessere Tools – werden eine entscheidende Rolle spielen. Der zweite Faktor ist fachliche Kompetenz, damit Java für immer höhere Leistungsanforderungen hochparalleler Systeme der nächsten Generation, große Heaps, große Datenmengen und fortgeschrittene Auswertungsverfahren gerüstet ist. Auch Cloud-Computing wird neue Anforderungen bringen wie High-Density-Deployment, Mandantenfähigkeit, Sicherheit in der Cloud und die Fähigkeit zur Anpassung an die durch Virtualisierung gewonnene Flexibilität.

John Duimovich

Programmieren in Java

Gelesen von Jürgen Thierack

„Viel Erfolg mit diesem Buch!“ wünscht der Autor, Prof. Fritz Jobst von der Hochschule Regensburg, in seinem Vorwort. In Besprechungen früherer Auflagen wurde im Internet darüber diskutiert, ob auch der absolute Anfänger, der sich zuvor noch mit keiner Programmiersprache befasst hat, mit diesem Buch ebenfalls „viel Erfolg“ haben kann. Man kann die Frage bejahen, denn das erste Kapitel „Der Einstieg in Java“ (12 Seiten) und das zweite Kapitel „Elemente der Programmierung“ (62 Seiten) sind so gehalten, dass es auch für den Erst-Einsteiger in die Welt der Programm-Erstellung genügend Erläuterungen gibt.

Für die Realisierung der Beispiele wird als Entwicklungsumgebung Eclipse benutzt, deren Funktion ausführlich mit vielen Bildschirm-Abbildungen vorgestellt ist.

Der Autor versteht sein Buch „als Gegenentwurf zu den Java-Bibeln“. Gegenüber der zweiten Auflage sind dreißig Seiten hinzugekommen. Mit seinen nunmehr fast vierhundert Seiten ist das Buch schon recht umfangreich, was sofort die Frage aufwirft, wie tief es nach dem elementaren Einstieg in die Java-Welt hineingeht. Fortgeschrittene Kapitel behandeln Grafik-Anwendungen, Multi-Threading, Programmierung in Netzwerken, Anbindung von Datenbanken mit JDBC sowie die Bearbeitung von XML.

Der Versuch, bestimmte Begriffe einzudeutschen, bringt „Programmfäden“ (Threads), „Verklemmungen“ (Deadlocks), Zusicherungen“ (Assertions) und „Sammlungen“ (Collections) hervor. Umgekehrt in der Reihenfolge der Benutzung des deutschen und englischen Begriffs, heißt es im Vorwort „Zur Verwaltung von Objekten dienen die sogenannten Collections“, während der Abschnitt 4.2, der den Collections gewidmet ist, „Verwalten von Objekten mit Sammlungen“ heißt, nachdem im Kapitel 3 ein weiterer deutscher Begriff für Collection eingeführt wurde: „Verwaltungsobjekte“. Das ist zwar eine originelle Wortschöpfung, jedoch nicht gerade eine treffende Übersetzung von „Collection“. Eine zwischen Übernahme der angelsächsischen Begriffe und Übersetzungen ins Deutsche hin und her pendelnde Begrifflichkeit kann gerade Anfänger verwirren.

Fazit: Trotz Unstimmigkeiten zwischen Inhaltsverzeichnis und den Seiten, wo der Inhalt dann tatsächlich steht, ist das Buch vom Fachlichen her hervorragend. Die Beispiele sind gut und ausführlich erklärt.

Autor:	Fritz Jobst
Titel:	Programmieren in Java
Verlag:	Hanser München
Umfang:	393 Seiten
Preis:	29,90 Euro
ISBN:	978-3-446-41771-7

eBook (downloadbar) im Preis enthalten

Jürgen Thierack
thierack@igfm-muenchen.de

Android: von Layouts und Locations

Andreas Flügge, Object Systems GmbH

Der Einstieg in eine komplett neue Programmier- und Laufzeit-Umgebung ist manchmal nicht ganz einfach. Bis eine Anwendung zum Laufen gebracht wird, sind neben dem Umgang mit der API auch Kenntnisse zur Konfiguration und zum Umgang mit der Entwicklungs-Umgebung notwendig.

Um den Einstieg in die Applikationsentwicklung mit Android zu demonstrieren, zeigt der Artikel eine bewusst einfach gehaltene Beispiel-Anwendung. Das Gerüst der vom Wizard generierten Anwendung wird anschließend erweitert und die gewünschte Funktionalität hinzugefügt.

MyLocation

Für einen ersten Überblick wird zunächst festgehalten, was die zu entwickelnde Applikation „MyLocation" leisten soll. Im Wesentlichen besteht sie aus einer View, die den aktuellen Status des GPS-Empfängers in Form der zuletzt festgestellten Geo-Koordinaten darstellt. Das klingt nicht besonders anspruchsvoll, aber auf dem Weg zur lauffähigen Anwendung gilt es, die eine oder andere Hürde zu überwinden. Ausgangspunkt ist das Code-Gerüst, das mit dem Wizard des Android Software Development Kit (SDK) erstellt wurde (siehe Java aktuell, Ausgabe 01/2012).

Layout und Views

Das Layout von Android-Anwendungen, das zunächst das grundsätzliche Aussehen einer Anwendung bestimmt, wird standardmäßig in XML-Dateien gespeichert. Der Wizard hat beim Anlegen unserer Anwendung bereits das Default-Layout erstellt, es liegt in der Datei „main.xml" im Verzeichnis „res/layout". Für die Bearbeitung des Layouts stellt das Android Software Development Kit einen grafischen Editor zur Verfügung. Alternativ lassen sich die XML-Dateien natürlich auch mit jedem beliebigen Text-Editor bearbeiten.

Das Layout soll einfach gehalten werden, alle Elemente sind untereinander dargestellt. Der Umgang mit dem Editor bedarf einer gewissen Einarbeitung, die Bedeutung der einzelnen Elemente und Attribute findet man auf den Referenzseiten des SDK. Für unsere Applikation bedienen wir uns des „LinearLayout", in das wir den Applikationsnamen, die zwei Titel- und Werttexte sowie den Button platzieren. Der Button wird zunächst ohne Funktion bleiben, kann aber später für Erweiterungen genutzt werden.

Mit den Attributen der einzelnen Elemente sorgen wir für die linksbündige Ausrichtung der Label, die rechtsbündige Ausrichtung der Werte und die Zentrierung von Applikationsnamen und Button. Das dazugehörige Layout wird im Editor, wie in Abbildung 1 zu sehen ist, dargestellt. Die XML-Darstellung entspricht dem Listing 1.

Die dort referenzierten Texte befinden sich in der Datei „strings.xml" im Verzeichnis „res/values" (siehe Listing 2). Diese können ebenfalls mit dem im SDK enthaltenen Editor oder auch textuell bearbeitet werden. Will man die Anwendung später internationalisieren, muss lediglich eine sprachspezifische Version dieser Datei erstellt und dann in einer leicht modifizierten Verzeichnisstruktur („res/values-<länderkürzel>") abgelegt werden. Das Android-Framework sorgt anhand der Spracheinstellungen des Geräts dann für die Verwendung der passenden Datei.

Beim Starten der Applikation in Eclipse (MyLocation -> Run As -> Android Application) wird die Anwendung im Emulator installiert und gestartet. Übersetzt und gebaut wird sie automatisch mit den Standardeinstellungen bei jeder Änderung. Wir sehen die erstellte View auf dem virtuellen Gerät. Diese sieht noch reichlich unspektakulär aus, denn es werden nur die Default-Daten angezeigt und hinter dem Button liegt keine Funktionalität.

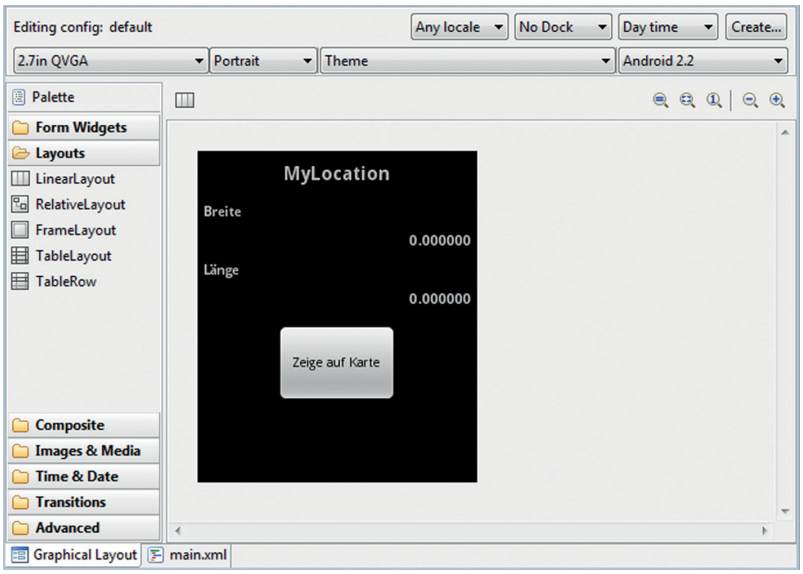


Abbildung 1: Layout im grafischen Editor

Standortbestimmung

Im nächsten Schritt möchten wir erreichen, dass die Anzeige automatisch aktualisiert wird, sobald sich der momentane Standort verändert. Android stellt zu diesem Zweck

einige Klassen zur Verfügung, die das Auslesen des GPS-Device ermöglichen und die Applikation darüber informieren können.

Im Package „android.location" befinden sich die zur Standortbestimmung benötigten Klassen. Von zentraler Bedeutung ist hier der „LocationManager". Dieser ist in der Lage, vom System einen Dienst anzufordern, der uns Geo-Informationen liefern kann. Diese Klasse („LocationManager") wird allerdings nicht direkt instanziiert. Die Methode „getSystemService(Context.LOCATION_SERVICE)" liefert eine solche Instanz, die wir unter anderem dazu benutzen können, uns über veränderte Geopositionen informieren zu lassen.

Grundlage dafür ist, dass wir eine Klasse mit dem Interface „LocationListener" implementieren lassen, die über die Methode „onLocationChanged" die Informationen über die veränderte Position verarbeiten kann. Das Interface verlangt die Implementierung weiterer Methoden, die in unserem Fall aber ohne Funktionalität bleiben können. Der Einfachheit halber machen wir dies direkt mit unserer Haupt-Activity „MyLocation".

Die Methode „onLocationChanged" in unserem Bespiel aktualisiert lediglich die Werte („Texte") in der View. Sie stellt mithilfe der Methode „findViewById" die Views der Koordinatenwerte in der „Activity" fest und ändert ihre Inhalte entsprechend. Jetzt fehlt nur noch die Registrierung unserer „Activity" als „LocationListener". Dazu nutzen wir die beiden Lebenszyklus-Methoden „onResume" und „onPause". Wir erinnern uns: Wird eine „Activity" sichtbar, so wird deren Methode „onResume" aufgerufen und sie wechselt in den Status „aktiv". Wird sie hingegen durch andere Komponenten verdeckt, wechselt sie in den Status „inaktiv", nachdem die Methode „onPause" passiert wurde. Praktisch bedeutet dies, dass wir in unserer „Activity" beide Methoden überschreiben und die Registrierung bzw. Deregistrierung unseres Listeners in der jeweiligen Methode durchführen.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:orientation="vertical"
   android:layout_height="wrap_content">
        <TextView
           android:layout_height="wrap_content"
           android:layout_margin="10px"
           android:textSize="8pt"
           android:text="@string/app_name"
           android:layout_width="wrap_content"
           android:gravity="center_horizontal"
           android:layout_gravity="center_horizontal"
           android:textStyle="bold"
           android:id="@+id/TitleTextView"/>
<TextView
           android:layout_height="wrap_content"
           android:layout_margin="5px"
           android:textSize="6pt"
           android:text="@string/breiteString"
           android:layout_width="wrap_content"
                android:layout_gravity="left"
           android:textStyle="bold"
           android:id="@+id/BreiteTextView"/>
...
        <Button
                android:id="@+id/KarteButton"
                android:text="@string/karteString"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center_horizontal"
                android:layout_margin="10px">
        </Button>
</LinearLayout>
```

Listing 1: Layout in main.xml

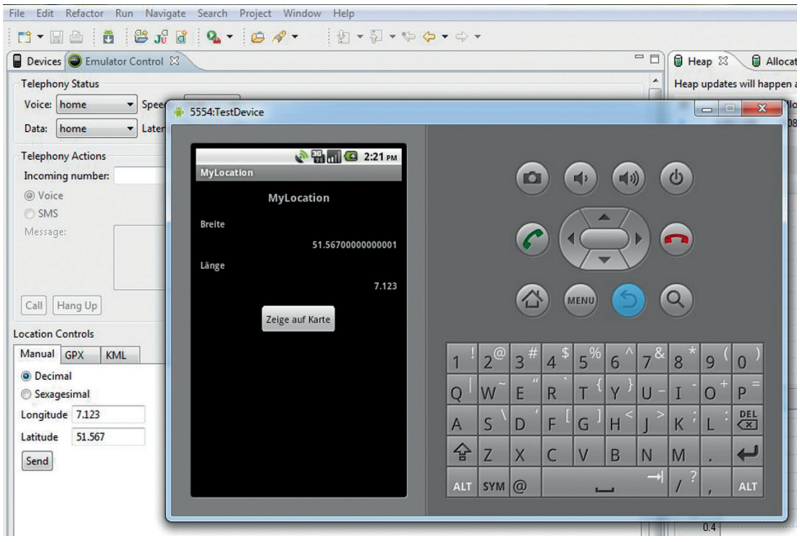


Abbildung 2: Simulierte GPS-Koordinaten

Energiespar-Aspekte

Die Klasse „LocationManager" bietet dazu die beiden Methoden „requestLocationUpdates" und „removeUpdates" an. Durch dieses Vorgehen garantieren wir ein möglichst energieeffizientes Arbeiten unserer Applikation. Die Daten werden nur dann

angefordert und ausgewertet, wenn die Anwendung auch wirklich aktiv ist.

Die Registrierungsmethode verlangt als Parameter den Typ des Location Providers (neben dem GPS-Empfänger kann es noch weitere geben), den zu registrierenden Listener sowie Angaben (Zeit und Entfernung) zum Intervall der Aktualisierung. Diese letzten beiden Informationen haben ebenfalls Auswirkungen auf den Energieverbrauch und sollten je nach Anwendungsfall geeignet gewählt werden. In unserem Beispiel wird der Einfachheit halber eine ständige Aktualisierung parametrisiert, die aber auch einem maximalen Energiebedarf entspricht. Listing 3 zeigt den Code für unsere Activity.

Berechtigung

Um die Applikation starten zu können, ist jetzt noch ein letzter Schritt notwendig. Durch die Nutzung der Location-API haben wir unserer Anwendung eine Fähigkeit verliehen, die eine besondere Berechtigung verlangt, nämlich die Nutzung des GPS-Empfängers. Das Android-Sicherheitskonzept kennt ungefähr hundert verschiedene Berechtigungen, die dem Anwender zeigen, welche Aktivitäten eine zu installierende Applikation auf seinem Gerät ausführen darf. Jede dieser Berechtigungen wird vom Entwickler im Manifest der Anwendung registriert. Vergisst er dies, beendet sich die Anwendung, sobald eine entsprechende Funktion im Code verwendet wird, mit einer Exception.

Das Manifest ist ebenfalls eine XML-Datei. Das SDK bringt aber auch hierfür einen eigenen Editor mit. In unserem Fall müssen wir die Berechtigung „ACCESS_FINE_LOCATION“ hinzufügen. Listing 4 zeigt das entstandene Manifest.

Test im Emulator

Für einen ersten Test benötigen wir kein reales Android-Device, der Emulator hat die Möglichkeit, für unsere Anwendung GPS-Koordinaten zu simulieren. Dazu wechseln wir nach dem Start der Applikation in die DDMS-Perspektive (Dalvik Debug Monitor Service). Dort befinden sich im linken unteren Bereich des Emulator-Control-Reiters die Location Controls. Hier eingegebene Koordinaten werden nach Betätigung des Send-Buttons dem virtuellen Device als neue Position übermittelt. Wie erwartet, werden daraufhin die Felder unserer Activity aktualisiert (siehe Abbildung 2).

```
<?xml version=”1.0” encoding=”utf-8”?>
<resources>
    <string name=”app_name”>MyLocation</string>
    <string name=”laengeString”>Länge</string>
    <string name=”breiteString”>Breite</string>
    <string name=”laengeWert”>0.000000</string>
    <string name=”breiteWert”>0.000000</string>
    <string name=”karteString”>Zeige auf Karte</string>
</resources>
```

Listing 2: Texte in strings.xml

```
public class MyLocation extends Activity implements LocationListener {
    LocationManager locationManager;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        locationManager=(LocationManager)getSystemService(
            Context.LOCATION_SERVICE);
    }
    @Override
    protected void onPause() {
        super.onPause();
        locationManager.removeUpdates(this);
    }
    @Override
    protected void onResume() {
        super.onResume();
        locationManager.requestLocationUpdates(
            LocationManager.GPS_PROVIDER, 0, 0, this);
    }
    @Override
    public void onLocationChanged(Location location) {
        TextView breiteView = (TextView)findViewById(
            R.id.BreiteWertView);
        TextView laengeView = (TextView)findViewById(
            R.id.LaengeWertView);
    breiteView.setText(String.valueOf(
        location.getLatitude()));
    laengeView.setText(String.valueOf(
        location.getLongitude()));
    }
    /* unwichtige Methoden ausgelassen ... */
}
```

Listing 3: MyLocations-Activity

Fazit

Wir haben gesehen, wie eine Android-Applikation prinzipiell aufgebaut ist. Die Funktionsweise von Layouts und Views sowie das Auslesen des GPS-Empfängers haben wir ebenso betrachtet wie die Auswirkungen der Nutzung von sicherheitsrelevanten Funktionen auf das Manifest. Wir konnten aufzeigen, dass wir mit relativ wenig Source-Code in kurzer Zeit eine voll funktionsfähige Anwendung erstellen können und diese auch ohne physisches Gerät trotz hardwarenaher Funktionen sehr gut testen konnten. Es können natürlich nicht alle Aspekte einer Android-Applikation beleuchtet werden. Aber mit der entwickelten Anwendung haben wir eine Basis gelegt, die hoffentlich zum weiteren Experimentieren mit Android anregt.

Links

- Downloadseite des Android Developers: http://developer.android.com/sdk/index.html
- Installation des Android SDK: http://developer.android.com/sdk/installing.html

Andreas Flügge
info@ordix.de

Andreas Flügge ist seit 2002 Senior Consultant bei der Object Systems GmbH, einer Tochtergesellschaft der ORDIX AG. Seit 1999 ist er im Java-Umfeld tätig. Seine Schwerpunkte sind Java-Enterprise-Umgebungen und Software-Architekturen.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.objectsystems.mylocation"
    android:versionCode="1"
    android:versionName="1.0">
  <uses-sdk android:minSdkVersion="8" />
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></
uses-permission>
  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".MyLocation"
              android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Listing 4: AndroidManifest.xml

Der iJUG im Java Community Process

Oliver Szymanski, Java User Group Erlangen

Im Jahr 1998 wurde Java Version 1.2 veröffentlicht und der Java Community Process (JCP) ins Leben gerufen, über den per Java Specification Requests (JSR) Änderungen an Java vorgeschlagen und in die Java-Technologie integriert werden können. Seit Kurzem arbeitet der Interessenverbund der Java User Groups e. V. (iJUG) im JCP mit.

Über den JCP sind auch Unternehmen wie Apple, Siemens, IBM oder Hewlett-Packard in die Weiterentwicklung von Java involviert. Über öffentliche Mailinglisten kann jeder den Lebenszyklus eines JSR verfolgen, auch ohne aktiv daran mitzuarbeiten. Ziel des iJUG ist die umfassende Vertretung der gemeinsamen Interessen der Java-Usergroups. Dies bauen wir nun mit einem engagierten Team weiter aus und übernehmen Verantwortung im Java Community Process. Als erste Schritte nimmt der iJUG bereits im JSF und JPA vertreten durch Dr. Bernd Müller aktiv teil und betreut diese Java Specification Requests. Da der iJUG insbesondere ein Augenmerk auf die Zukunft von Java selbst hat, ist momentan die Teilnahme an JSR 308 (Annotations), JSR 335 (Lambda) sowie JSR zu Java Platform Enterprise Edition in Planung.

Bei allen Feature-Wünschen, Anforderungen und dergleichen, die wir aus der Community erhalten und versuchen in die jeweiligen JSR einzubringen, muss natürlich beachtet werden, dass sie für die Community und nicht für einzelne interessant sind. Dies wird sicherlich in Zukunft zu Diskussionen führen und macht deutlich, wie wichtig Feedback aus der Community, also den Java-Usergroups ist.

Das iJUG-Team besteht momentan aus Andreas Badelt (DOAG Deutsche ORACLE-Anwendergruppe e.V.), Markus Eisele (DOAG Deutsche ORACLE-Anwendergruppe e.V.), Michael Hüttermann (Java User Group Köln), Bernd Müller (Ostfalia) und Oliver Szymanski (Java User Group Erlangen-Nürnberg). Das Team nimmt gerne Anregungen und Vorschläge entgegen.

Oliver Szymanski
oliver.szymanski@source-knights.com

Oliver Szymanski (Dipl. Inform., Univ.) ist als Software-Architekt / Entwickler, Berater und Trainer in den Bereichen Java, .NET und Mobile-Development tätig. Parallel dazu arbeitet er als Schriftsteller.

UI-Entwicklung mit JavaServer Faces und CDI

Andy Bosch, www.jsf-academy.com

Mit Java EE 6 haben sowohl JSF 2.0 als auch CDI 1.0 Einzug gehalten. Es ist jedoch nicht unbedingt auf den ersten Blick erkennbar, dass sich beide Standards in idealer Weise ergänzen. JSF fokussiert sich klar auf den UI-Layer, während CDI eher neutral zwischen den Layern anzusiedeln ist. Doch CDI ist weit mehr als lediglich Glue-Code, um (Software-)Komponenten über Schichtengrenzen hinweg zu verbinden. Es liefert eine Menge von Funktionen, die in JSF fehlen, durch CDI aber perfekt ergänzt werden können. Der Artikel gibt einen Überblick und Einblick, wie CDI JSF sinnvoll erweitert.

```
@ManagedBean
@SessionScoped
public class JsfPerson {

        private String firstname;
        private String lastname;

        @ManagedProperty(value=„#{jsfShoppingCart}“)
        private JsfShoppingCart cart;

        // setter und getter
}

@ManagedBean
@SessionScoped
public class JsfShoppingCart {
  ...
}
```

Listing 1: Managed Beans und deren Injection in JSF

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>
    javax.faces.webapp.FacesServlet
  </servlet-class>
  <load-on-startup> I </load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
<context-param>
  <param-name>javax.faces.PROJECT_STAGE</param-
name>
  <param-value>Development</param-value>
</context-param>
<listener>
  <listener-class>
    org.jboss.weld.environment.servlet.Listener
  </listener-class>
</listener>
```

Listing 2: Notwendige Konfiguration in der web.xml

Grau ist alle Theorie. Deswegen stürzen wir uns auch gleich in die Praxis und bauen eine Web-Anwendung mit JSF und CDI auf. Alle Features werden an kurzen Sourcecode-Beispielen erläutert, sodass man dies auch in seinem eigenen Workspace nachvollziehen kann. Basis ist eine Standard-JSF-2.0-Anwendung. In diese wird CDI integriert; anschließend werden die verschiedenen Möglichkeiten der Kombination betrachtet. Um möglichst leichtgewichtig zu sein, ist die Anwendung für einen Apache-Tomcat–Servlet-Container konfiguriert. Sie wäre aber auch ohne größere Aufwände in einem beliebigen Standard-Java-EE-6-Container lauffähig.

Worauf JSF zielt und was CDI bietet

JSF existiert seit 2004; die ersten Anfänge in der Expert Group gehen bis auf das Jahr 2001 zurück. Wichtig ist jedoch die Einordnung von JSF als ein reines UI-Framework. Der Fokus ist somit klar gegeben. Alles rund um den Presentation-Layer wird durch JSF abgehandelt, während Fragen nach Persistenz, Business-Logik oder Remote Calls durch diese Spezifikation nicht beantwortet werden.

Spricht man von „Komponenten“ in JSF, sind damit die UI-Komponenten gemeint, also beispielsweise eine Eingabe-, Button- oder Tabellen-Komponente. Dahinter liegt das Modell in Form von JSF Managed Beans, in denen die Werte aus der Oberfläche übertragen werden.

CDI dagegen lässt sich nicht eindeutig einer Schicht zuordnen. Vielmehr kann mit CDI eine lose Kopplung von Software-Komponenten erfolgen. Diese wird über Dependency Injection realisiert. Das bedeutet, dass Beans, die durch die CDI Runtime gemanagt werden, in andere CDI-Beans injiziert werden können.

CDI-Beans weisen (analog zu JSF) einen dedizierten Lifecycle vor und können in einem Context leben. Rund um die Themen „Kontexte“ und „Dependency Injection“ hat die CDI-Spezifikation viele zusätzliche Funktionen bereitgestellt. Der Fokus liegt hier also nicht auf einem Layer, sondern auf der Verbindung von Software-Komponenten, egal in welche Schicht diese einzuordnen sind.

DI in JSF

JSF kennt ein eigenes Bean-Management-Konzept. Beans können entweder über die bekannte „faces-config.xml“ oder mittlerweile auch über Annotationen als „managed“ deklariert werden. Dabei wird den Beans ein Scope zugewiesen. Auch in JSF gibt es einen einfachen Dependency-Injection-Mechanismus (siehe Listing 1).

Das Konzept der JSF Managed Beans ist für einfache Anwendungen durchaus einsetzbar, weist allerdings auch einige Einschränkungen auf. Das JSF-eigene Bean-Konzept verfügt über einen eigenen Lifecycle und ist nicht mit anderen Komponenten-Modellen abgestimmt. Die Injection erfolgt leider nicht typsicher, sodass man hier gegebenenfalls erst zur Laufzeit Fehlermeldungen erhält. Kleines Detail am Rande: Klassen wie die obigen sollten in der Praxis natürlich nicht mit „Jsf...“ beginnen, dies dient lediglich der Illustration.

Vom UI- in den Service-Layer

Doch wie erfolgt nun der Aufruf eines fachlichen Service aus einer JSF Managed Bean

heraus? Hier liefert die JSF-Spezifikation keine Antwort, da man hier den UI-Layer eben verlässt. Man konnte sich bislang eigene Konstrukte überlegen oder mit Frameworks wie Spring arbeiten.

Ganz anders bei CDI, das sich als allgemeines Komponenten-Modell positioniert, wobei unter dem Begriff „Komponente" in diesem Zusammenhang etwas abstrakter eine Software-Komponente gemeint ist. Man spricht im Umfeld von CDI analog zu JSF auch von „Beans", allerdings diesmal von CDI-Beans. CDI weist ebenfalls einen Dependency-Injection-Mechanismus auf, der jedoch im Gegensatz zu JSF typsicher und weitaus flexibler ist. CDI-Beans sind ebenfalls, wie bei JSF, kontextbehaftet und leben in einem dedizierten Lifecycle. Bei den Scopes, die CDI mitbringt, existieren neben den üblichen Kandidaten wie „RequestScope" oder „SessionScope" auch einige sehr interessante Scopes wie der in der Vergangenheit oft herbeigesehnte „ConversationScope". Da CDI nicht speziell einer Schicht zugeordnet werden kann, könnte man damit sowohl UI-Beans als auch Service-Beans verwalten und somit einen Aufruf aus dem UI in den Service ermöglichen.

Projekt-Setup

Angenommen, man arbeitet mit einem Java-EE6-Application-Server. Dann wäre das Thema des Projekt-Setups schon erledigt, denn sowohl JSF als auch CDI sind Teil des Java-EE-Stacks und müssen damit nicht mehr separat bereitgestellt werden. In dem Fall, dass ein Servlet-Container zum Einsatz kommt, sind sowohl JSF als auch CDI separat zu deployen. Für die gezeigten Beispiele kommen die aktuellste JSF-Referenz-Implementierung (Mojarra) sowie die CDI-Referenz-Implementierung (JBoss Weld) zum Einsatz. Nachdem die notwendigen Libraries in das WEB-INF/lib-Verzeichnis aufgenommen sind, muss man lediglich Ergänzungen in der „web.xml" vornehmen sowie eine weitere XML-Datei anlegen (siehe Listing 2).

Für JSF muss dazu das zentrale „FacesServlet" sowie ein „FacesMapping" angegeben werden. Das Setzen der „ProjectStage" empfiehlt sich auf jeden Fall, um bessere Hinweise während der Entwicklung zu erhalten. Für „Weld" ist im Umfeld eines Servlet-Containers noch der Eintrag eines „ServletListeners" notwendig.

Ganz wichtig ist jetzt noch, im „web-inf"-Verzeichnis eine (leere) „beans.xml"-Datei anzulegen. Am Vorhandensein dieser Datei erkennt die CDI-Runtime, dass sich das Scannen der Klassen lohnt, um darin CDI-Beans zu finden. Fehlt diese Datei, findet die Runtime diese nicht.

Injection-Varianten

Zunächst folgt ein sehr simples Beispiel, um CDI-Beans in JSF zu verwenden (siehe Listing 3). In diesem ersten Beispiel sind bereits die wichtigsten Grundlagen für CDI zu erkennen. Die Annotation „@Named" bewirkt, dass die CDI-Bean mit einem Namen in der XHTML-Seite ansprechbar ist. Diese Annotation wäre aus CDI-Sicht nicht zwingend notwendig. Wenn man jedoch eine CDI-Bean in einer XHTML-Seite ansprechen will, ist dieser ein Name zu geben. Genau das bewirkt die „@Named"-Annotation.

Die Angabe „@SessionScoped" ist an sich selbsterklärend. Sie bewirkt, dass die Bean in der Session angesiedelt ist und so lange Bestand hat, wie die Session gültig ist. Interessant wird es bei der Annotation „@Inject" auf der Property „player". Hier wird eine passende CDI-Bean vom Typ „Person" gesucht, die dann per CDI-Magic in das entsprechende Game-Objekt injiziert wird. Die Injection kann dabei auch mittels Interfaces funktionieren. Angenommen, die Klasse „Person" wäre ein Interface (siehe Listing 4).

Wenn ein Interface-Typ bei der Injection verwendet wird, sucht die CDI-Runtime nach einer passenden Implementierung für das „Person"-Interface. Vorausgesetzt, dass diese Suche auf ein eindeutiges Ergebnis stößt, erfolgt wiederum eine entsprechende Injection.

Listing 5 zeigt das Ansprechen einer CDI-Bean. Da „Game" mit „@Named" annotiert ist, wird per Konvention über den Namen „game" (Klassenname und den ersten Buchstaben davon klein schreiben) und die bekannte Punktnotation auf die Eigenschaft „nameOfTheGame" zugegriffen. Die Punktnotation funktioniert auch über die injizierte Eigenschaft „player".

Qualifier

Bislang sind wir davon ausgegangen, dass die CDI-Runtime die Injection immer eindeutig vornehmen kann. Wir haben

```
@Named
@SessionScoped
public class Game {

  private String nameOfTheGame;

  @Inject
  private Person player;

  // setter and getter
  ...
}
```

Listing 3: CDI-Bean mit „@Named"-Annotation

```
public interface Person {
        public String getFirstname();
        public void setFirstname(String firstname);
        public String getLastname();
        public void setLastname(String lastname);
}
```

Listing 4: „Person"-Interface

```
<h:body>
  JSF+CDI Beispiel
  <br />
  Name of the game:
    <h:outputText value=„#{game.nameOfTheGame}" />
  Name of the player:
    <h:outputText value=„#{game.player.firstname}" />
</h:body>
```

Listing 5: Ansprechen von CDI-Beans in XHTML

```
@Qualifier
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE, ElementType.FIELD})
public @interface Volljaehrig {}
```

Listing 6: Festlegen eines Qualifiers

```
public class Game {
  @Inject @Volljaehrig
  private Person player;
  ..
```

Listing 7: Injection Point mit Qualifier

```
@Statistics
@Interceptor
public class StatisticsAspect {
  @AroundInvoke
  public Object writeStatistics(
        InvocationContext ctx ) throws Exception {
    System.out.println( „Methodenaufruf von: „ +
        ctx.getMethod().getName() );
    return ctx.proceed();
  }
}
```

Listing 8: Statistik-Interceptor

```
@InterceptorBinding
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD, ElementType.TYPE})
public @interface Statistics {
}
```

Listing 9: Interceptor-Binding

```
@Statistics
public void someCoolFunction() {
   ...
}
```

Listing 10: Verwendung der Statistik-Annotation

```
@Named
@RequestScoped
public class GameController {

        @Inject
        private Conversation conv;

        public void startGame( SystemEvent evt ) {
                if ( conv.isTransient() ) {
                        conv.begin();
                }
        }
}
```

Listing 11: Starten einer Conversation

```
<h:body>
   JSF+CDI Beispiel
        <br />
        Sie möchten ein neues Spiel beginnen?
        Kann klicken sie <h:link value=„hier“
outcome=„gameplay“ />
</h:body>
```

Listing 12: GET-Request auslösen in JSF

gesehen, dass wir bei der Injection sogar Interface-Typen verwenden können und CDI automatisch nach einer passenden Implementierung sucht. Doch was passiert, wenn es mehrere Implementierungen gibt? Konkret auf unser Beispiel des Interface „Person“ bezogen: Angenommen, es gibt zwei Klassen, „NichtVoll-jaehrige-Person“ und „VolljaehrigePerson“. Eventuell gibt es noch weitere Klassen wie „Kind“ oder „Mitarbeiter“, die ebenfalls das Interface implementieren. Welche Implementierung soll CDI nun für die Injection verwenden? Da dies nicht mehr eindeutig ist, wird beim Start der Anwendung eine entsprechende Exception geworfen. Wir müssen somit an dieser Stelle eine genauere Angabe vornehmen, damit CDI wieder allein weiterarbeiten kann. Die Antwort hierauf heißt „Qualifiers“ (siehe Listing 6). Dort ist eine neue Annotation aufgebaut (konkret ein Qualifier) mit dem Namen „VolljaehrigePerson“. Diesen Qualifier kann man an der konkreten Klasse (z.B. auch an der Klasse „Mitarbeiter“) einsetzen und am Injection Point zusätzlich mit angeben (siehe Listing 7).

Damit kann die CDI-Runtime wieder eine eindeutige Zuordnung finden – vorausgesetzt natürlich, es gibt nicht zwei Klassen, die das „Person“-Interface implementieren und beide mit „@Volljaehrig“ annotiert sind.

Interceptoren

Mit Interceptoren aus CDI lassen sich hervorragend sogenannte „orthogonale Funktionalitäten“ abbilden. Damit sind Funktionen gemeint, die nicht direkt etwas mit der fachlichen Umsetzung eines Use-Case zu tun haben, sondern eher technische Aspekte darstellen, die an verschiedenen Stellen im Programm benötigt werden. Genau das war immer das Argument für Aspekt-orientierte Programmierung (AOP). Vielen war (und ist teilweise immer noch) AOP zu komplex, undurchsichtig oder einfach nur suspekt. Doch bietet AOP eine Möglichkeit, die in Anwendungen häufig vernachlässigt wird: Regelmäßig wiederkehrende orthogonale Funktionalität an eine zentrale Stelle herauszuziehen (einen Aspekt eben) und damit den Rest des Sourcecodes übersichtlicher zu gestalten.

CDI-Interceptoren sind die Antwort aus CDI-Sicht. Natürlich sind diese kein Ersatz für AOP; sie liefern dennoch einen hilfreichen Ausschnitt aus der Welt von AOP, mit dem schon sehr viele Möglichkeiten gegeben sind. Vorab sei noch erwähnt, dass Interceptoren keine komplett neue Errungenschaft von CDI sind. Auch ohne CDI lassen sich Interceptoren in Java EE nutzen. CDI liefert hierbei allerdings ein paar nette Ergänzungen mit.

Konkret kann mittels eines Interceptors auf eine Methoden-Ausführung Zugriff genommen werden. Es kann ein sogenannter „Around Advice“ abgebildet werden, mit dessen Hilfe zum Beispiel vor einer jeden Methoden-Ausführung ein bestimmter Code ausgeführt wird (siehe Listing 8). Dort ist ein einfacher Statistik-Aspekt abgebildet. Die Methode „writeStatistics“ soll vor jedem (relevanten) Methoden-Aufruf ausgeführt werden. Hier könnte beispielsweise ein Statistik-Modul angebunden werden. In Listing 8 wird lediglich ein „System.out“ angestoßen. Für diesen Interceptor ist des Weiteren ein Interceptor-Binding notwendig (siehe Listing 9). Ausgestattet mit dem Interceptor und dem Interceptor-Binding lässt sich anschließend eine Methode entsprechend annotieren (siehe Listing 10).

Arbeiten mit Conversations

Ein Highlight, das CDI für JSF-Entwickler bereithält, ist der „ConversationScope“. Damit hat ein JSF-Entwickler endlich die Möglichkeit, mittels Standard-Mechanismen einen Scope zu haben, der länger ist als ein Request, aber zugleich kürzer als eine Session. Zudem ist es heutzutage schon fast die Regel, dass Benutzer einer Web-Anwendung im „Multi-Tabbing“-Modus arbeiten, also mit mehreren Fenstern oder Tabs gleichzeitig auf derselben Web-Applikation agieren. Dies kann eine Web-Anwendung durchaus in starke Bedrängnis bringen. Denn in welchem Scope sollen Beans gespeichert werden, die in verschiedenen parallelen Fenstern bearbeitet werden? „SessionScope“ würde bedeuten, dass alle Fenster auf die gleichen Daten zugreifen und somit auf die gleichen Objekte schreiben wollen. Dies ist nicht immer gewollt. Der „RequestScope“ ist häufig zu kurz, da Daten über mehrere Requests hinweg gehalten werden sollen.

Die Conversation ist hier die perfekte Antwort. Jedes Fenster/Tab bekommt seine eigene Conversation, sodass darin unabhängig voneinander gearbeitet wer-

den kann. Auf Modellseite muss dazu eine Bean lediglich mit „@ConversationScoped“ annotiert sein. Interessant ist, wie eine Conversation gestartet wird, wenn ein neues Tab geöffnet wird. Nehmen wir als Anwendungsfall wieder das Umfeld eines Spiels (siehe Listing 11).

Dort sieht man zunächst, dass der GameController eine CDI-Bean ist, die durch die „@Named“-Annotation auch wieder über eine XHTML-Seite ansprechbar ist. Da der Controller in der Regel „stateless“ ist, genügt die „@RequestScoped“-Angabe. Neu kommt hinzu, dass man sich die aktuelle Conversation injizieren kann. Dies erfolgt über „@Inject“ und den Typ „Conversation“. Damit kann programmatisch zum passenden Zeitpunkt auf dem Conversation-Objekt die „begin“-Methode aufgerufen werden. Doch wie kann man erreichen, dass die Methode „startGame“ beim Öffnen eines neuen Tabs oder Fensters angesprungen wird? Hier kommt wieder JSF ins Spiel (siehe Listing 12).

Seit JSF 2.0 gibt es ein neues Tag „<h:link>“, mit dem ein GET-Request ausgelöst werden kann. War JSF früher lediglich auf „post“ beschränkt, erzeugt das „<h:link>“-Tag nun eine „get“-URL, die durch „Open-in-new-tab“ vom Benutzer aus aufgerufen werden kann. Somit haben wir schon einmal den ersten Teil erreicht, der Benutzer kann von der Game-Hauptseite aus verschiedene konkrete Games in neuen Tabs öffnen. Jetzt müssen wir nur noch erreichen, dass in der eigentlichen Game-Seite eine Conversation gestartet wird, also die Methode „startGame“ in unserem Controller aufgerufen wird. Auch hier greifen wir auf ein neues JSF-Feature zu: „SystemEvents“, und dort konkret „PreRenderViewEvent“.

```
<f:event listener=„#{gameController.start-
Game}“
        type=„preRenderView“></f:event>
```

„PreRenderView“ wird ausgelöst, kurz bevor die Seite gerendert wird. Damit erreichen wir, dass für jedes separate Fenster/Tab eine eigene Conversation gestartet wird und alle Beans, die mit „@ConversationScoped“ annotiert sind, parallel, aber unabhängig voneinander bestehen können.

Fazit

In der Kombination von JSF und CDI lassen sich hervorragende Anwendungs-Designs umsetzen. Die Funktionen aus CDI integrieren sich ideal in JSF-Anwendungen. Ein weiterer Vorteil ist zudem, dass CDI durchaus als leichtgewichtig bezeichnet werden und ohne großen Overhead sehr einfach und schnell in JSF integriert werden kann. Auch die aktuellen Aktivitäten der JSF Expert Group, die momentan an der Version 2.2 arbeitet, deuten auf eine verstärkte Integration von JSF und CDI hin.

Andy Bosch
andy.bosch@jsf-academy.com

Andy Bosch arbeitet als selbständiger Trainer und Berater für JSF und Portlets. Im Rahmen seiner Initiative unter www.jsf-academy.com hat er ein Trainingsportal rund um JSF und JSF-nahe Technologien aufgebaut. Er hält regelmässig Vorträge auf nationalen und internationalen Konferenzen und ist Autor verschiedener Bücher und Fachartikel.

Einstieg in Java 7

Gelesen von Jürgen Thierack

Der Autor, über den wir auf dem Umschlag erfahren, dass er Senior-Software-Architekt ist, behauptet im Vorwort gleich zu Anfang: „Dieses Buch ist eines der meistverkauften Java-Bücher Deutschlands geworden.“

Im Unterschied zu „Programmieren in Java“ (siehe Seite 17) hat Bernhard Steppan in seinem Werk Ausführlichkeit und Breite bei den Grundlagen zum Leitfaden erhoben. Dabei geht es nicht nur um Java, sondern auch um Allgemein- und Basiswissen zu Programmieren und Computer-Hardware. Es werden die Sprachen der ersten bis sechsten Generation vorgestellt, nachdem die Basis der elektronischen Informationsverarbeitung in Form von Bits, Bytes, Zahlensysteme, Kodierung von Zeichen erarbeitet wurde. Grundlegendes, wie ein Computer überhaupt funktioniert, erfährt man im Anhang.

Wer sich bei Galileo Press registriert, kann sich den Quellcode herunterladen. Das Buch gibt Anleitungen, wie die Beispiele in Eclipse und NetBeans zu importieren sind.

Fazit: Natürlich wendet sich das Buch an den Anfänger. Teil III („Größere Java-Projekte“) ist aber auch für Leute geeignet, die schon Grundwissen in Java haben. Dennoch ist dem Fortgeschrittenen eher zu Prof. Jobsts „Programmieren in Java“ zu raten (siehe Seite 17).

Jürgen Thierack
thierack@igfm-muenchen.de

Autor:	Bernhard Steppan
Titel:	Einstieg in Java 7
Verlag:	Galileo Press
Umfang:	605 Seiten
Preis:	19,90 Euro
ISBN:	978-3-8362-1662-3

Testzugang zu einer Online-Fassung im Preis enthalten

JSFUnit

Bernd Müller und Boris Wickner, Ostfalia, Hochschule für angewandte Wissenschaften

JSFUnit ist ein Werkzeug für den Komponenten- und Integrationstest von JSF-Anwendungen. Es basiert auf Arquillian und ist wie Arquillian ein JBoss-Projekt. Der Artikel charakterisiert JSFUnit und stellt den Einsatz beispielhaft vor.

JSFUnit wird bereits seit mehreren Jahren für Komponenten- und Integrationstests von JSF-Anwendungen eingesetzt. Erste Quellen im Internet datieren von 2007. JSFUnit basierte auf Cactus [1], einem Apache-Projekt, das die Möglichkeit von Komponententests für serverseitigen Code wie Servlets und EJBs zum Ziel hatte. Cactus wird mittlerweile nicht mehr weiterentwickelt und ist in den Attic-Bereich von Apache gezogen. Durch diese Entwicklung von Cactus, vor allem aber durch das neue, hausinterne JBoss-Projekt Arquillian [2], das zukünftig einen zentralen Platz bei den Test-Werkzeugen von JBoss einnehmen wird, stand eine Überarbeitung von JSFUnit an. Die aktuelle und von den Autoren verwendete Version ist 2.0.0.Beta2.

```
<h:form id="form">
 Vorname: <h:inputText id="vorname"
          value="#{kundeView.vorname}" /> <br/>
 Nachname: <h:inputText id="nachname"
          value="#{kundeView.nachname}" /> <br/>
 <h:commandButton id="speichern"
    action="#{kundeView.speichern}"
    value="Speichern" />
</h:form>
```

Listing 1

```
@ManagedBean
public class KundeView {

  private String vorname;
  private String nachname;

  public KundeView() { }

  public String speichern() {
    return "speichern-ok.xhtml";
  }
...
}
```

Listing 2

```
<h:outputText id="ok" value="Kunde #{kundeView.
vorname}
          #{kundeView.nachname} gespeichert" />
```

Listing 3

JSFUnit-Tests werden im Container ausgeführt und erlauben den vollständigen und transparenten Zugriff auf JSF-Komponenten wie Managed Beans, EL-Ausdrücke, UI-Komponenten und die JSF-Konfiguration. Durch die Möglichkeit, einerseits clientseitige Benutzereingaben zu simulieren sowie andererseits auf JSF Managed Beans und über Arquillian auf EJBs zugreifen zu können, ist das Erstellen von Integrationstests über alle Schichten einer Java-EE-Anwendung leicht möglich.

Die Beispiel-Anwendung

Um das Ganze nicht zu abstrakt zu halten, entwickeln wir eine Anwendung, die Kundendaten persistiert. Wir verwenden JBoss AS 7, da dieser als JBoss-Produkt die beste Unterstützung durch JSFUnit und Arquillian genießt. Beginnen wir mit der JSF-Seite, die wie alle folgenden Code-Beispiele möglichst einfach gehalten ist und deren wichtigste Zeilen im Folgenden dargestellt sind (siehe Listing 1).

Es fällt zunächst die Verwendung des „id"-Attributs bei allen JSF-Tags auf. Um die interne Funktionsweise von JSF zu garantieren, muss jede Komponente eine eindeutige Id besitzen. Wird diese nicht explizit angegeben, generiert JSF eine solche Id. Diese ist jedoch wesentlich schwieriger in JSFUnit zu verwenden, sodass beim Einsatz von JSFUnit die Grundregel gilt, dass jede Komponente eine explizite Id erhalten sollte. Die für die JSF-Seite erzeugte HTML-Seite enthält diese Ids ebenfalls und das von JSFUnit intern verwendete HtmlUnit [3] nutzt diese, um Daten einzugeben und auszulesen sowie um Formulare abzuschicken.

Kommen wir nun zur verwendeten Managed Bean „KundeView". Diese ist zunächst ein einfaches POJO mit der @ManagedBean-Annotation (siehe Listing 2).

Das Navigationsziel, die JSF-Seite „speichern-ok.xhtml", enthält als einfache Nachricht an den Benutzer die Bestätigung der Kundenspeicherung (siehe Listing 3).

Mit den beiden JSF-Seiten und der Managed Bean sind wir nun in der Lage, unseren ersten JSFUnit-Test zu schreiben. Wir wählen JUnit als Testwerkzeug und geben Arquillian als eigentlichen Test-Runner mit der Annotation „@RunWith" an (siehe Listing 4). Für eine Einführung in Arquillian verweisen wir auf den Artikel von Frederik Mortensen in der Java aktuell 4/2011 [4].

Mit „@Deployment" wird die Methode annotiert, die das Deployment-Archiv erzeugt. Arquillian baut hier auf Shrinkwrap [5] auf, ebenfalls ein JBoss-Projekt. Es werden der Reihe nach definiert: die Art des Archivs (hier ein WAR), die zu verwendende web.xml, die zu deployenden Packages, die beiden JSF-Seiten sowie die „faces-config.xml", die in diesem Fall als leere Datei erzeugt wird. Wir verwenden ein von Eclipse erzeugtes Dynamic-Web-Project, was die Verwendung von „WebContent" als Präfix der JSF-Seiten erklärt. Auch die Benutzung verschiedener Verzeichnisse

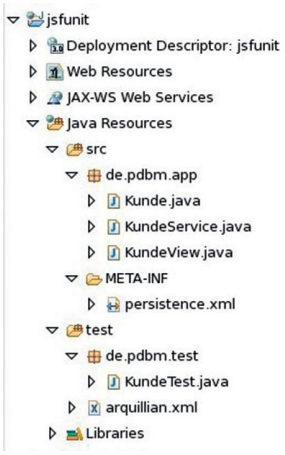


Abbildung 1: Ausschnitt des Eclipse Project Explorers

für die Anwendung und die Tests sind kein Problem. Abbildung 1 zeigt den entsprechenden Ausschnitt des Project Explorers von Eclipse. Die noch nicht verwendeten Klassen und die „persistence.xml" kommen im nächsten Beispiel dazu. Der eigentliche Test wird durch JUnits @Test-Annotation markiert (siehe Listing 5).

Die initiale Seite ist mit der Annotation „@InitialPage" definiert. Die Test-Methode bekommt als Parameter ein „JSFServerSession"- und ein „JSFClientSession"-Objekt übergeben, die die entsprechenden Sessions repräsentieren. Mit „client.setValue()" erfolgt eine Benutzereingabe in die entsprechende Komponente. Wir haben zu Demonstrationszwecken beide Versionen, einmal mit und einmal ohne Formular-Präfix, gewählt. Die Methode „client.click()" schickt das Formular ab. Mit „server.getManagedBeanValue()" kann ein EL-Ausdruck evaluiert werden. „client.getElement()" liefert die entsprechende HTML-Komponente in der Antwort.

Beispiel-Anwendung mit EJB und JPA

Wir erweitern nun unsere Anwendung, um die Persistenz der Kundendaten zu realisieren. Zunächst machen wir aus der JSF-Bean eine CDI-Bean, da CDI auch für die Injektion der EJB verwendet wird. Wir geben nur die geänderten Teile der Bean wieder (siehe Listing 6).

Das JPA-Entity ist ebenfalls sehr minimalistisch gehalten, bekommt aber eine Named-Query, um alle Kunden zu selektieren (siehe Listing 7).

Die EJB besitzt Methoden, um einen Kunden zu speichern und alle Kunden zu lesen (siehe Listing 8).

Damit die CDI-Bean und die Injektion korrekt verwendet werden, muss das Deployment den CDI-Deployment-Deskriptor „beans.xml" enthalten. Wir verzichten auf eine erneute Darstellung der Deployment-Methode. Ein Integrationstest mit Eingabe der Kundendaten, Abschicken des Formulars, Einfügen in die Datenbank und anschließendem Prüfen der Datenbank-Inhalte ist nun leicht möglich. Wir beschränken uns hier auf einen rein quantitativen Test. Da wir in der „persistence.xml" Hibernate anweisen, alle Tabellen neu zu erzeugen, können wir von einer leeren Kundentabelle ausgehen. Nach dem Test muss daher genau ein Kunde existieren (siehe Listing 9).

Weitere Möglichkeiten von JSFUnit

Die Autoren konnten in diesem Artikel die Möglichkeiten von JSFUnit nur anreißen. Mit JSFUnit können praktisch alle für JSF relevanten Konstrukte getestet werden. Auch der anspruchsvolle Umgang mit generierten Ids für ein „<h:dataTable>" ist möglich, wie sie in ihrem JSF-Buch [6] für JSFUnit 1.2 gezeigt haben.

JSFUnit 2.0 bietet neben der verwendeten „@InitialPage"-Annotation eine Reihe weiterer Annotationen an, um die Erstellung von Tests zu erleichtern. Zu diesen gehören „@BasicAuthentication", „@FormAutentication", „@Proxy", „@Browser" und „@Cookies" mit der jeweiligen offensichtlichen Bedeutung.

Wo Licht ist, ist auch Schatten

Es wurde bisher nur beschrieben, was mit JSFUnit möglich ist. JSFUnit ist jedoch im Augenblick noch in einem Zustand, der auch einiges an Kritik erlaubt. Derzeit existiert keine Distribution für JSFUnit, sondern lediglich die Möglichkeit einer Maven-basierten Installation. Diese resultiert in 151 Jars mit einer Größe von insgesamt 41 MB. Dies erscheint den Autoren nicht angemessen.

Nach dem Wechsel von JBoss 7.0.1.Final auf 7.0.2.Final misslangen alle Tests ohne erkennbaren Grund. Die Verwendung von GlassFish 3.1.1 scheiterte komplett. Der Zugriff auf CDI-Beans über „server.getManagedBeanValue()" ist ebenfalls nicht

```
@RunWith(Arquillian.class)
public class KundeTest {

  @Deployment
  public static WebArchive createDeployment() {
    WebArchive war = ShrinkWrap
      .create(WebArchive.class, „test.war“)
      .setWebXML(new File(„WebContent/WEB-INF/
web.xml“))
      .addPackage(Package.getPackage(„de.pdbm.app“))
      .addPackage(Package.getPackage(„de.pdbm.test“))
      .addAsWebResource(new File(„WebContent/kunde.
xhtml“))
      .addAsWebResource(
        new File(„WebContent/speichern-ok.xhtml“))
      .addAsWebInfResource(EmptyAsset.INSTANCE,
                    „faces-config.xml“);
    return war;
  }
...
}
```

Listing 4

```
@Test
@InitialPage(„/kunde.jsf“)
public void testKunde(JSFServerSession server,
                JSFClientSession client) {
  String vorname = “Max”;
  String nachname = “Mustermann”;
Assert.assertEquals(“/kunde.xhtml”,
                server.getCurrentViewID());
  client.setValue(„form:vorname“, vorname);
  client.setValue(„nachname“, nachname);
  client.click(„speichern“);
  Assert.assertEquals(vorname,
    server.getManagedBeanValue(„#{kundeView.vorname}“));
  Assert.assertEquals(nachname,
    server.getManagedBeanValue(„#{kundeView.nachname}“));
  Assert.assertEquals(„Kunde „ + vorname + „ „
                + nachname + “ gespeichert”,
      client.getElement(“ok”).getTextContent());
  Assert.assertEquals(“/speichern-ok.xhtml”,
       server.getCurrentViewID());
  }
```

Listing 5

```
@Named
@RequestScoped
public class KundeView {
  ...  @Inject
  KundeService kundeService;
  ...
  public String speichern() {
    kundeService.speichern(new Kunde(vorname,
     nachname));
    return “speichern-ok.xhtml”;
  }
  ...
}
```

Listing 6

```
@Entity
@NamedQuery(name = „Kunde.alleKunden“,
        query = „SELECT k from Kunde k“)
public class Kunde {

        @Id @GeneratedValue
        private Integer id;
        private String vorname;
        private String nachname;
  ...
}
```

Listing 7

```
@Stateless
public class KundeService {

  @PersistenceContext
  EntityManager em;

  public void speichern(Kunde kunde) {
  em.persist(kunde);
  }

  public List<Kunde> alleKunden() {
    return em.createNamedQuery(“Kunde.alleKunden”,
          Kunde.class).getResultList();
  }
}
```

Listing 8

```
...
Assert.assertEquals(0, kundeService.alleKunden().size());
client.setValue(„vorname“, vorname);
client.setValue(„nachname“, nachname);
client.click(“speichern”);
Assert.assertEquals(1, kundeService.alleKunden().size());
...
```

Listing 9

möglich. Hier können nur JSF-Managed-Beans verwendet werden, obwohl dies nach Foren-Einträgen auch für CDI-Beans möglich sein sollte. Abschließend ist noch die sehr rudimentäre Dokumentation zu bemängeln.

Fazit

JSFUnit wird zurzeit intern auf Arquillian und ein annotationsbasiertes API umgestellt. Die von uns verwendete Version 2.0.0.Beta2 empfiehlt sich im Augenblick noch nicht für den produktiven Einsatz. Wir hoffen, dass es JBoss im finalen Release gelingen wird, das Qualitätsniveau früherer Releases zu erreichen.

Weiterführende Informationen

[1] http://jakarta.apache.org/cactus
[2] http://www.jboss.org/arquillian
[3] http://htmlunit.sourceforge.net
[4] Frederik Mortensen. Arquillian, Java aktuell 4/2011
[5] http://www.jboss.org/shrinkwrap
[6] Bernd Müller, JavaServer Faces 2.0, Hanser, 2010

Bernd Müller
bernd.mueller@ostfalia.de
Boris Wickner
bo.wickner@ostfalia.de

Bernd Müller ist Professor für Software-Technik an der Ostfalia. Er ist Autor des Buches „JavaServer Faces 2.0“ und Mitglied in der Expertengruppe des JSR 344 (JSF 2.2).

Boris Wickner ist Mitarbeiter an der Ostfalia im Bereich Infrastruktur für die Software-Ausbildung und Master-Student an der TU Braunschweig.

Vorschau

Java aktuell – Herbst 2012
Die nächste Ausgabe erscheint am 6. Juni 2012.
Falls Sie einen Artikel darin veröffentlichen möchten, schicken Sie bitte vorab Ihren Themenvorschlag an redaktion@ijug.eu

Redaktionsschluss ist am 4. April 2012

UML lernen leicht gemacht – welche Editoren sich am besten eignen

Andy Transchel, Universität Duisburg-Essen

Es war einmal eine Idee … diese Idee wurde von fünfzehn Studenten der Universität Duisburg-Essen geteilt. Sie machten es sich zur Aufgabe, im Rahmen eines Praxisprojekts des Studiengangs „Angewandte Kognitions- und Medienwissenschaften" UML-Editoren zu evaluieren. Zudem haben sie einen eigenen Editor programmiert, um einen guten Einstieg in die Unified Modeling Language zu bieten. Dieser soll Schüler und Anfänger das Erstellen von UML-Diagrammen auf einfache, jedoch effektive Art und Weise lehren.

Das Praxisprojekt beschäftigt sich mit der Frage, welcher kostenfreie UML-Editor sowohl für Schüler und Anfänger als auch für die Lehrkraft besonders gut geeignet ist. Bei der Untersuchung dieser Frage liegt der Schwerpunkt nicht darauf, den besten Editor zu finden, sondern die Unterschiede mehrerer hervorzuheben und diese miteinander zu vergleichen. Zudem ist es von großer Bedeutung, Vorteile und Defizite einzelner Editoren aufzuzeigen.

Ein Grund für die Evaluation von ausschließlich kostenfreien UML-Editoren liegt unter anderem in der Zielgruppe des eigenen Editors: Schüler und allgemein interessierte Anfänger sollen sich beim Erstellen von UML-Diagrammen zunächst einen Überblick über wichtige Funktionen verschaffen können. Dafür ist eine kostenpflichtige Software nicht zwingend notwendig. Editoren, für die man kein Geld aufwenden muss, können einen guten Einstieg in die Welt der Unified Modeling Language bieten. Sie müssen lediglich gut verständlich und anfängerfreundlich sein. Teure Programme bieten oftmals schon zu detaillierte und für einen Anfänger zu komplizierte Funktionen an, die einen ersten Umgang mit UML unnötig erschweren könnten.

Für die Evaluation wurden zehn UML-Editoren ausgesucht. Diese bieten eine große Resonanz sowie einen schnellen Erfolg bei der Suche nach einem UML-Editor im Internet.

Eine grundlegende und wichtige Frage, die es jedoch zunächst zu beantworten gilt, ist, welche Kriterien man für die Evaluation von UML-Editoren mit einbeziehen und untersuchen möchte. Hierbei muss man eine genaue Vorstellung davon haben, welche Funktionen wichtig für diese Art von Software sind und was generell beachtet werden sollte, um einen Editor benutzerfreundlich zu gestalten. Erst im Anschluss daran kann man aus diesen Kriterien Fragen und Anforderungen ableiten und im Verlauf der Evaluation alle Editoren daraufhin untersuchen und bewerten.

Bei der Auswahl orientierte sich die Gruppe an die Software-ergonomischen Prinzipien der Dialoggestaltung nach der ISO Norm 9241. Diese Norm trägt seit 2006 den Titel „Ergonomie der Mensch-System-Interaktion" (siehe Hansjörg Rampl, 2007, Handbuch Usability unter: http://www.handbuch-usability.de/iso-9241.html) und beschreibt Qualitätsrichtlinien zur Sicherstellung der Ergonomie interaktiver Systeme. In einem Abschnitt über die Grundsätze der Dialoggestaltung schlägt sie sieben Kriterien vor, welche sich auch für die Evaluation von UML-Editoren ideal eignen: Aufgabenangemessenheit, Erlernbarkeit, Selbstbeschreibungsfähigkeit, Erwartungskonformität, Steuerbarkeit, Individualisierbarkeit sowie Fehlertoleranz. Zudem entschied man sich noch für ein weiteres, achtes Kriterium, das Design des Editors.

Jeder Editor wurde auf alle Kriterien hin untersucht und anschließend ausgewertet. Dazu wurden für jede zu untersuchende Aufgabenstellung Teilnoten vergeben, woraus sich für jeden Editor schließlich eine Gesamtnote errechnen ließ. Besonderes Augenmerk wurde bei der Evaluation auf die Aufgaben-Angemessenheit, die Erwartungskonformität und die Fehlerbehandlung gelegt. Diese drei Prinzipien sollten beim Erlernen des Umgangs mit UML-Diagrammen die größte Beachtung finden. So flossen sie mit einem höheren Prozentanteil in die Auswertung der Evaluation ein.

Die Ergebnisse waren größtenteils negativer als erhofft, nur ein Editor konnte sich das Prädikat „gut" verdienen (ArgoUML: Ø 2,4). Der Artikel beleuchtet drei der zehn Editoren näher.

ArgoUML

Mit einer Gesamtnote von 2,4 ist ArgoUML der Testsieger der Evaluation. Der UML-Editor konnte sich besonders im Bereich der Fehlerbehandlung von den anderen Programmen abheben und bekam hier die beste Note unter allen Tools. Dabei überzeugte der Editor durch eine Checkliste, die den Erwartungen an ein Fehlerprotokoll entsprach. Diese Checkliste deckt alle Fragen ab, die bei einer Aktion während des Modellierens aufkommen können. Um sein Diagramm auf etwaige Ungereimtheiten zu überprüfen, kann man hinter alle

gestellten Fragen ein Häkchen machen. Bleibt eine Frage ohne Häkchen, so weiß man für sich selbst, dass das Diagramm noch bearbeitet werden muss, damit es hinterher logisch und sinnvoll ist. Zudem werden im Diagramm selbst auch Fehler oder noch nicht bearbeitete Aspekte rot markiert, sodass man auch hier das Feedback bekommt, dass das Diagramm noch mal überarbeitet werden sollte. Vergleichbares war bei keinem anderen geprüften Tool zu finden.

Leider werden jedoch nicht alle Fehler erkannt, was den Lerneffekt senkt. Es reichte jedoch für die Bestnote in diesem Bereich. Weniger gut schnitt die Software hingegen bei der Erfüllung des Prinzips der Individualisierbarkeit ab. Zwar ist es möglich, zwischen verschiedenen Sprachen zu wählen, was schon bei der Installation des Programms als Erstes positiv auffällt, allerdings lässt sich das Programm nicht an die Bedürfnisse eines Anfängers anpassen, etwa durch das Zurateziehen eines Hilfsassistenten.

Bezüglich der Selbstbeschreibungsfähigkeit gab es zwar gute Ansätze, wie beispielsweise die Vorgabe von Default-Werten beim Erstellen eines Klassen-Diagramms, diese sind aber leider nicht gut umgesetzt und wirken eher verwirrend. Positiv war dagegen, dass beim Erstellen von Klassen verschiedene Eingabefelder geöffnet werden, über welche man die Beziehungen und Eigenschaften steuern kann. Davon profitieren vor allem Anfänger.

Auch im Bereich der Aufgabenangemessenheit zeigte ArgoUML an vielen Stellen zumindest im Ansatz Einfallsreichtum. So wurden verschiedene Diagramm-Typen implementiert, für die auch Schablonen vorgegeben sind, was ein schnelles Arbeiten ermöglichen soll. Jedoch wird gerade dieses erschwert, da bei vielen Änderungen an diesen Schablonen unnötig viele Schritte zur Zielerreichung notwendig sind. Beim Kriterium der Erlernbarkeit konnte sich der Editor zwar erneut von den anderen Tools abheben, allerdings nicht in einem überzeugenden Ausmaß. Hierbei lässt nämlich die angebotene Hilfefunktion etwas zu wünschen übrig, da zu einigen wichtigen Themen noch gar keine Anleitung verfasst wurde, obwohl diese schon im Inhaltsverzeichnis der Hilfe erwähnt werden. Hier wäre eine baldige Ergänzung äußerst wünschenswert. Abschließend lässt sich jedoch trotzdem sagen, dass ArgoUML ein recht ordentliches Tool ist, das viele Eigenschaften und Bestandteile in sich vereint, die für die Anwendung als Lernhilfe benötigt werden. An einigen Stellen sind jedoch leider noch unnötige Interaktionsschritte vorhanden und verschiedene Funktionen wurden nicht ideal umgesetzt. Dennoch ist diese Software, die in der Version 0.32.2 getestet wurde, ein verdienter Testsieger.

StarUML

StarUML heißt ein weiterer Editor, der bei der Evaluation genau untersucht wurde. Dieser überzeugt zwar hinsichtlich vieler nützlicher Nebenfunktionen, allerdings lässt er in seiner Hauptaufgabe, nämlich dem Erstellen von UML-Diagrammen, die Leichtigkeit im Umgang vermissen. Dies führte am Ende dazu, dass er im Mittelmaß landete und nur mit einer befriedigenden Note (3,1) bewertet wurde.

Beim Start von StarUML wird dem Nutzer direkt das erste Zurechtfinden erleichtert. Es wird ein Fenster geöffnet und man kann sich entscheiden, ob man ein neues Diagramm erstellen oder eine bereits bestehende Datei öffnen möchte. Entscheidet man sich für ersteres, bleibt noch die Wahl des Diagramm-Typs. Hierbei könnte es bei einem unerfahrenen Nutzer das erste Mal zu Schwierigkeiten kommen, da ihm ein „4+1 View Model“ wahrscheinlich unbekannt vorkommt.

Klickt man auf die zweite Auswahlmöglichkeit „UML Components Approach“, weil ja zumindest der bekannte Begriff „UML“ darin vorkommt, stehen einem nur die Elemente eines Use-Case-Diagramms zur Verfügung. Also sollte man einfach dieses sich zu Beginn öffnende Fenster schließen. Denn dann bekommt man zumindest Diagramm-Elemente vorgesetzt, um ein Klassen-Diagramm erstellen zu können. Bisher gibt es hierfür also bereits Abzüge bezüglich der Selbstbeschreibungsfähigkeit.
Des Weiteren fällt der Editor hinsichtlich der Fehlertoleranz auf ganzer Ebene durch. Man darf munter und ohne darüber nachzudenken modellieren, ohne dass Korrekturen durchgeführt werden oder Fehlermeldungen erscheinen.

Äußerst verwirrend stellt sich auch die gegebene Hilfefunktion dar. Diese ist eigentlich sehr übersichtlich aufgebaut, gut illustriert und durchaus anfängerfreundlich geschrieben, sodass der Nutzer sich hier viele nützliche Tipps holen kann. Allerdings ist dort von Diagramm-Typen die Rede, die im Programm selbst gar nicht implementiert wurden.

Eine weitere Frage stellt sich dann noch beim Modellieren: Warum werden beim Hinzufügen vieler Diagramm-Elemente automatisch chinesische Namen vergeben? Hierauf wird man wohl keine Antwort finden können, da es scheinbar einfach sinnfrei ist. Stattdessen wären, besonders zur besseren Unterscheidung der Elemente, neutrale Benennungen deutlich besser gewesen.

Positiv hervorzuheben ist dagegen das sehr gut erfüllte Prinzip der Steuerbarkeit. Man kann ausgeführte Schritte rückgängig machen, mit Shortcuts arbeiten, die Elemente sind leicht aus der Symbolleiste in die Zeichenfläche übertragbar und man kann sowohl einzelne Elemente als auch das ganze Diagramm verschieben, was nützlich ist, wenn dem Nutzer später auffällt, dass an einer Stelle des Diagramms, die jedoch weit am Rande der Zeichenfläche ist, noch Informationen fehlen.

Im Gesamten betrachtet bietet StarUML einiges, das richtig gut gelungen ist (erwartungskonforme Gestaltung, gute Steuerbarkeit, gute Funktionsvielfalt), aber leider auch vieles, das man hätte besser machen können, wie beispielsweise die Implementierung aller Diagramm-Typen, die die Hilfefunktion auch verspricht.

Anfänger müssen sich mit der sonst recht gelungenen Hilfe geduldig auseinandersetzen, dann kann man mit dieser Software einen Lernprozess in Gang setzen, der zu einem besseren Verständnis der Softwaremodellierung beiträgt.

Bouml

Bouml war mit der Gesamtnote 3,7 einer der beiden Verlierer dieser Evaluation. In der Steuerbarkeit konnte dieses Tool noch eine überzeugende Leistung erbringen. Hier war eine freie Gestaltung der Diagramme möglich und Icons wurden hilfreich eingesetzt.

Die Überprüfung der Erwartungskonformität ergab, dass das Programm für alle Betriebssysteme kompatibel ist und auch die Funktionen an den gewohnten

Abbildung 1: Das ist der tierische Assistent von PingUML

Stellen zu finden sind. In allen anderen Bereichen bekam Bouml, das in der Version 4.23 getestet wurde, jedoch eher schlechte Bewertungen. So konnte die Aufgabenangemessenheit nicht überzeugen. Zwar werden, bis auf Use-Case-Diagramme, die wichtigsten UML-Diagramm-Typen angeboten; der Weg, um ein solches zu erstellen oder zu ändern, ist jedoch viel zu lang und umständlich. Dadurch wird ein effizientes Arbeiten verhindert. Zudem wären – zur Unterstützung beim Erstellen – mehr Schablonen für die einzelnen Diagramm-Typen hilfreich gewesen. In einem ebenso schlechten Licht zeigte sich das Prinzip der Selbstbeschreibungsfähigkeit, denn ein intuitives Arbeiten wurde hier erschwert, da keine Defaultwerte verwendet wurden.

Das Tool ist anfangs sehr schwer zu durchschauen und ohne Unterstützung eher langsam erlernbar. Um dies auszugleichen, gibt es gute Tutorials, die aber nicht aktuell und dadurch auch nicht immer hilfreich waren. Dies führte deshalb zu einer negativen Bewertung der Erlernbarkeit. Zu bemängeln war auch, dass das Design nicht immer zur Funktionalität des Editors beitragen konnte. Verschiedene Elemente lenkten eher vom wichtigen Inhalt ab, und die Signalwirkungen der Farben hätten besser für diverse Funktionen genutzt werden können.

Am schlechtesten wurde die Individualisierbarkeit des Tools bewertet. Es gab keine Möglichkeiten, das Tool hinsichtlich der Sprache oder des eigenen Wissensstands anzupassen. Trotz aller Kritik muss aber gesagt werden, dass Syntaxfehler durch eine geschickte Fehlerüberprüfung vermieden werden und es sogar eine Korrektur von verschiedenen Fehlern gibt, die allerdings nicht immer funktioniert. Da jedoch keine weiteren Testkriterien an eine hilfreiche Fehlerbehandlung, wie beispielsweise ein Fehlerprotokoll, erfüllt wurden, konnten hier keine weiteren Punkte vergeben werden. Bouml ist in diesem Bereich dennoch einer der am besten benoteten Editoren. Letztendlich handelt es sich hier um ein Tool, das an vielen Stellen überarbeitet werden muss, um für den Einsatz in Schulen und Universitäten in Frage zu kommen.

Software-Modellierung leicht gemacht mit PingUML

Das zweite Ziel des Projekts war die Implementierung eines eigenen UML-Editors. Dieser sollte die Mängel, die bei der Evaluation zum Vorschein gekommen waren, beseitigen und besonders für unerfahrene Nutzer der Softwaremodellierung geeignet sein. Nach Wochen der akribischen Arbeit entstand PingUML, eine anfängerfreundliche Software, mit welcher der Einstieg in den Umgang mit UML erleichtert wird. Der Name des Editors entstand in Anlehnung an seinen Hilfsassistenten, einen kleinen Pinguin, dessen Geschlecht frei wählbar ist und der dem Nutzer jeden Modellierungsfehler aufzeigt (siehe Abbildung 1). So werden männliche wie auch weibliche Nutzer gleichermaßen dazu motiviert, sich näher mit PingUML auseinanderzusetzen und den Umgang mit dieser Software zu erlernen.

Der Prozess des Erlernens wird durch ein Tutorial unterstützt. Diese Hilfefunktion bietet eine Anleitung samt Screenshots zu allen durchführbaren Aktionen. So bekommt der Nutzer schnell einen Überblick über alle vorhandenen Funktionen und ein umfangreiches Know-how für das Erstellen von UML-Diagrammen (siehe Abbildung 2).

Ein weiteres Feature ist die Implementierung einer History. Diese zeigt dem Nutzer genau an, zu welchem Zeitpunkt er welchen Schritt ausgeführt hat. Klickt man auf einen Eintrag in der History, so springt man auch in der Zeichenfläche zu diesem Schritt zurück. So kann man hinterher jede einzelne Aktion genauestens nachvollziehen.

PingUML unterstützt vorläufig nur Klassen-Diagramme, wird aber in nachfolgenden Arbeiten sukzessive erweitert. Nichtsdestotrotz ist es ein hervorragender UML-Editor für den Einstieg in die Softwaremodellierung. Der Editor, der an der Universität Duisburg-Essen von fünfzehn Studierenden der angewandten Kognitions- und Medienwissenschaften und einer großartig unterstützenden Leitungskraft erarbeitet wurde, ist auf der Homepage http://www.compling.uni-due.de/pinguml zum kostenlosen Download bereitgestellt.

Andy Transchel
andy.transchel@stud.uni-due.de

Andy Transchel wurde 1989 in Herten geboren und machte dort am Städtischen Gymnasium Herten sein Abitur. Seit Oktober 2009 studiert er an der Universität Duisburg-Essen „Angewandte Kognitions- und Medienwissenschaften". Inzwischen lebt er in Düsseldorf und spielt dort neben dem Studium in einer Hobbymannschaft aktiv Fußball.

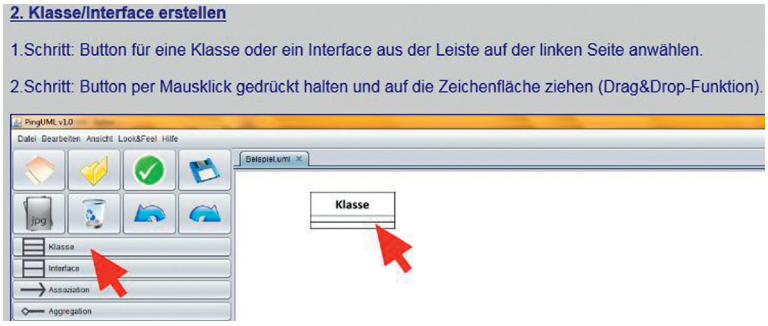


Abbildung 2: Auszug aus dem Tutorial zu PingUML

Webservices testen mit soapUI

Sebastian Steiner, Trivadis AG

soapUI ist ein populäres und sehr umfangreiches Testwerkzeug für serviceorientierte Lösungen. Dieser Artikel dient als Einführung in die Thematik und zeigt auf, welchen Mehrwert und welche Möglichkeiten soapUI bietet und wie man damit erfolgreich arbeitet.

Das Testen von SOA-Lösungen ist mindestens genauso wichtig wie das von klassischen Applikationen, da SOA-basierte Systeme oftmals das Rückgrat eines Unternehmens darstellen. Zudem werden sie von vielen Benutzern verwendet und kleine Fehler können gravierende Auswirkungen haben.

Wenn SOA-Lösungen getestet werden sollen, ergibt sich eine Vielzahl von Herausforderungen: Man hat es mit heterogenen Umgebungen zu tun, welche die unterschiedlichsten Anwendungen integrieren, und trifft oftmals auf asynchrone Systeme. Die Tests für diese Anwendungen sind jedoch auf eine synchrone Ausführung angewiesen: Der Test sendet eine Nachricht und erwartet sofort eine Antwort. Manchmal ist es für Komponenten in einer SOA-Umgebung nicht möglich, eine dedizierte Test-Umgebung bereitzustellen. Zudem ist es wichtig, dass Tests automatisiert und reproduzierbar ausgeführt werden können. Dazu ist man auf Test-Frameworks angewiesen. Abbildung 1 zeigt die Kern-Anforderungen für ein Test-Framework. Darauf kommt es beim Testen an:

1. Test-Daten generieren und damit die zu testende Komponente aufrufen können
2. Die Antwort, welche die Komponente zurückliefert, auf Richtigkeit validieren können
3. Gewisse Komponenten des Systems simulieren („mocken“) können

Was ist soapUI?

Die schwedische Firma eviware, die heute zu SmartBear Software gehört, hat mit soapUI eine sehr umfangreiche Test-Software auf den Markt gebracht. Der Hauptfokus von soapUI liegt auf dem Testen von SOAP-Webservices; dies ist auch der Kern dieses Artikels. soapUI bietet jedoch darüber hinaus noch zahlreiche weitere Funktionen an und unterstützt das Testen von REST-Services und Datenbanken. In Zusammenarbeit mit Hermes JMS (www.hermesjms.com) können auch JMS-Queues getestet werden. eviware selbst bezeichnet soapUI auch als „The Swiss Army Knife of Testing“.

soapUI ist in zwei unterschiedlichen Ausführungen verfügbar. Es gibt eine Open-Source- sowie eine kostenpflichtige Pro-Version. Letztere ist für rund 350 Dollar pro Jahr und Lizenz erhältlich und bietet dem Benutzer einen erweiterten Funktionsumfang, verschiedene Assistenten, welche die Bedienung vereinfachen, sowie Anrecht auf Support von eviware. In diesem Artikel wird später detaillierter auf die Eigenschaften der Pro-Version eingegangen.

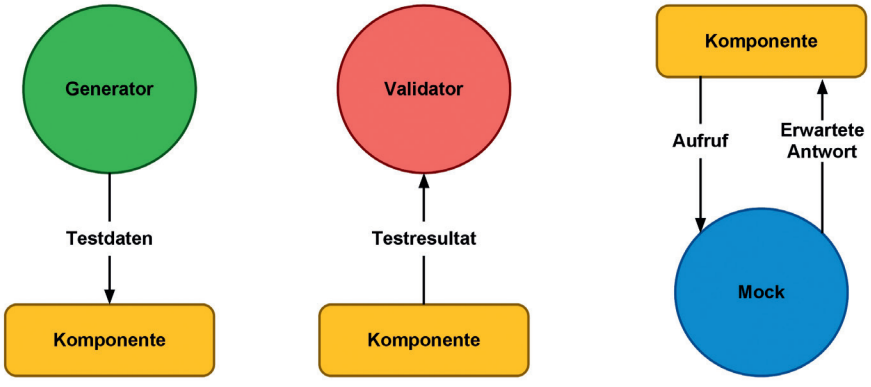


Abbildung 1: Kern-Anforderungen für ein Test-Framework

Erste Schritte

Erfreulich an soapUI ist der einfache Einstieg. Sobald man ein neues Projekt angelegt hat, fragt soapUI nach dem WSDL des zu testenden Webservice. Basierend darauf können gleich Testfälle mit Beispiel-Requests für alle Webservice-Operationen angelegt werden. Ein soapUI-Projekt besteht aus mehreren Testsuiten und diese aus mehreren Testfällen. Ein Testfall beinhaltet wiederum mehrere Schritte, sogenannte „TestSteps“. Die Testfälle kann man entsprechend anpassen, an den Server schicken und das Resultat überprüfen. Mit geringem Aufwand lässt sich somit ein Webservice testen. Abbildung 2 zeigt eine Übersicht des soapUI-Workspace.

In soapUI können auf unterschiedlichen Stufen (Projekt, Testsuite, Testfall) sogenannte „Properties“ definiert werden. Ein Property hat einen Namen und einen Wert. Diese Properties kann man den Attributen eines SOAP-Request zuweisen und diesen dadurch dynamisch gestalten. Listing 1 zeigt die Anwendung von Properties und demonstriert gleichzeitig auch, wie auf die Properties der unterschiedlichen Hierarchiestufen zugegriffen werden kann.

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.
org/soap/envelope/"
    xmlns:web="http://www.webservicex.
net">
   <soapenv:Header/>
   <soapenv:Body>
      <web:VerifyAddress>
         <web:City>${#Project#city}</
web:City>

<web:State>${#TestSuite#state}</
web:State>
         <web:Zip>${#TestCase#zip}</
web:Zip>
      </web:VerifyAddress>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 1: Ein SOAP-Request mit Properties

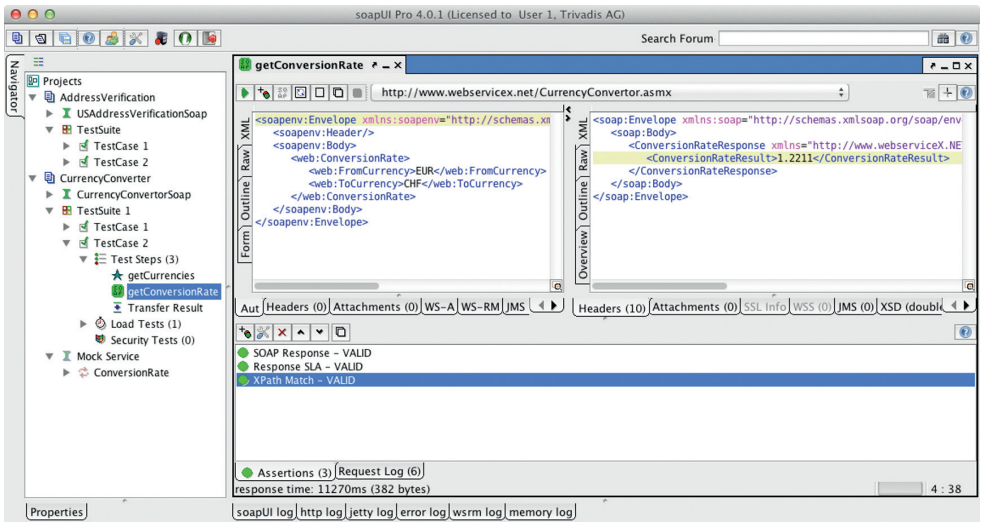


Abbildung 2: Der soapUI-Workspace

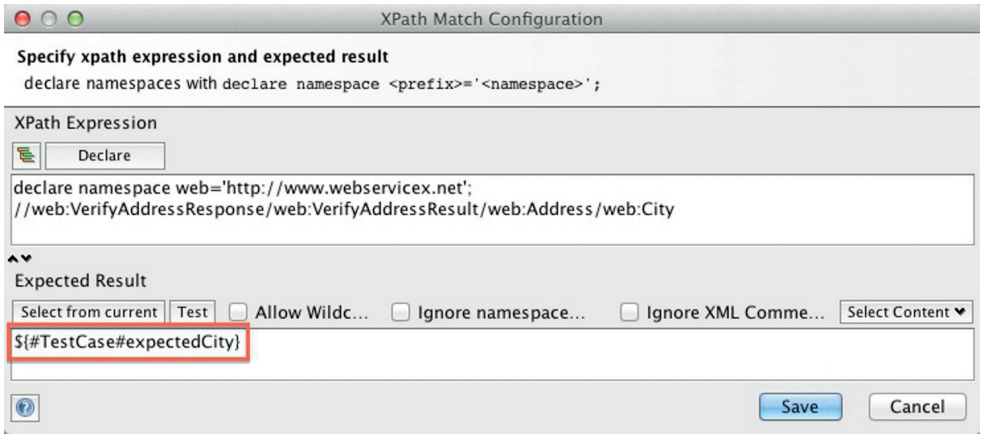


Abbildung 3: Definition einer XPath Assertion

Für komplexere Berechnungen oder die Implementierung einer Funktionalität, die soapUI nicht von Haus aus anbietet, gibt es den „Groovy TestStep". Man erhält mit diesem die Möglichkeit, sein eigenes Groovy-Skript als Teil eines Testfalls zu implementieren. Es stehen sämtliche APIs von JRE und soapUI zur Verfügung. Die soapUI-API ist auf der offiziellen Homepage dokumentiert (www.soapui.org/apidocs). Die Ausführlichkeit dieser Dokumentation ist jedoch beschränkt. Innerhalb des Skripts sind für den Zugriff auf das soapUI-Objektmodell standardmäßig kontextabhängige Variable definiert. Zudem hat man Zugriff auf Properties, Request- und Response-Daten von vorhergehenden Webservice-Aufrufen sowie Log-Objekte.

soapUI unterstützt „Set Up"- und „Tear Down"-Skripte. Diese können auf den drei Stufen „Projekt", „Testsuite" und „Testfall" definiert werden und kommen immer dann zur Ausführung, wenn das entsprechende Objekt hoch- oder heruntergefahren wird. Normalerweise wird dies verwendet, um Testdaten vorzubereiten oder um sie nach einem Test wieder zu löschen, beziehungsweise um Ressourcen freizugeben.

Validierung von Antwortdaten

Eine weitere Anforderung an ein Test-Framework ist wie bereits erwähnt die Validierung von Antwortdaten. Eine Validierung kann dabei funktional (wie „richtige Antwort") oder nicht-funktional (wie „Antwortzeit eingehalten") sein. Auf einem SOAP-Request lassen sich unterschiedliche Assertions definieren. Je nach Typ wird beispielsweise überprüft, ob die Antwort eine gültige SOAP-Antwort ist oder ob ein bestimmtes XML-Schema eingehalten wird. Eine SLA-Assertion überprüft das Einhalten einer vorgegebenen Antwortzeit. Eine mächtige und viel verwendete Assertion ist die XPath Assertion. Man definiert dabei einen Xpath-Ausdruck, der einen Knoten/Wert aus dem Antwortdokument zurückgibt. Die Assertion vergleicht diesen mit dem erwarteten Resultat. Dieses kann statisch sein oder aus einem Property ausgelesen werden, wie Abbildung 3 verdeutlicht. Es gibt weitere Assertions für verschiedene Anwendungsfälle.

Service Mocking

Die dritte Kernanforderung an das Test-Framework ist das Simulieren von Komponenten des zu testenden Systems. Damit lassen sich komplexe Prozesse simulieren, die nicht Bestandteil des Tests sind, wie beispielsweise das Aufrufen von externen Diensten oder Datenbankabfragen. Es können aber auch Komponenten des Systems simuliert werden, die sich noch in der Entwicklung befinden und noch gar nicht angesprochen werden können. In soapUI nennt man diesen Prozess „Service Mocking".

Abhängig von den Anforderungen können solche Mock-Services unterschiedlich komplex ausfallen. In der einfachsten Form haben sie pro Operation eine vordefinierte Antwort gespeichert, die immer dann

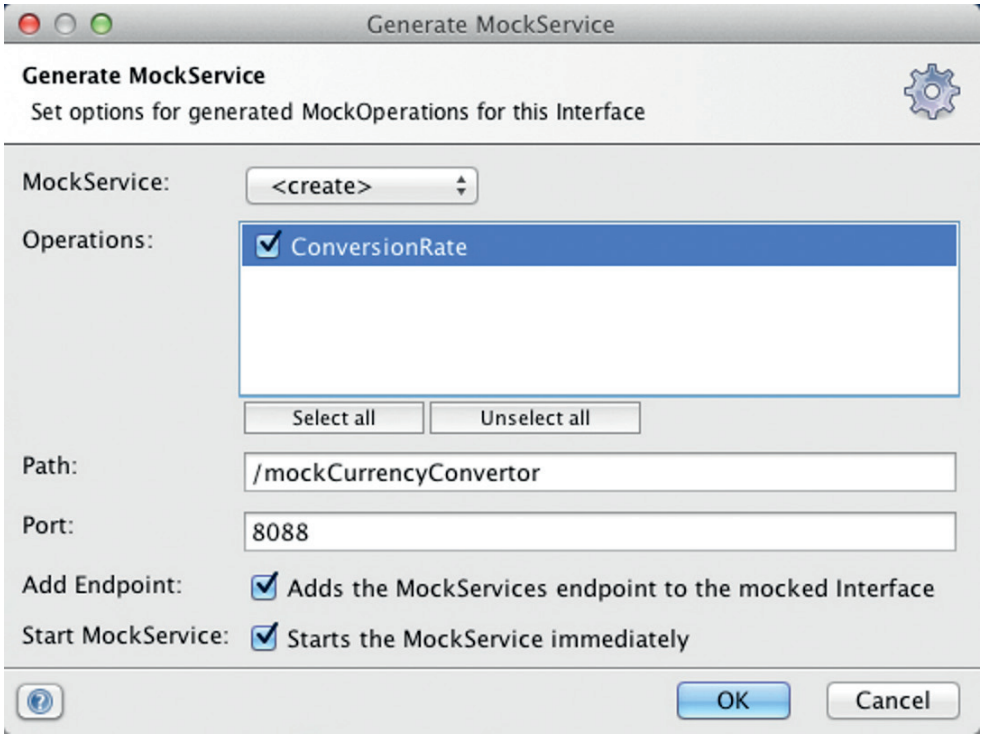


Abbildung 4: Anlegen und Starten eines Mock-Service

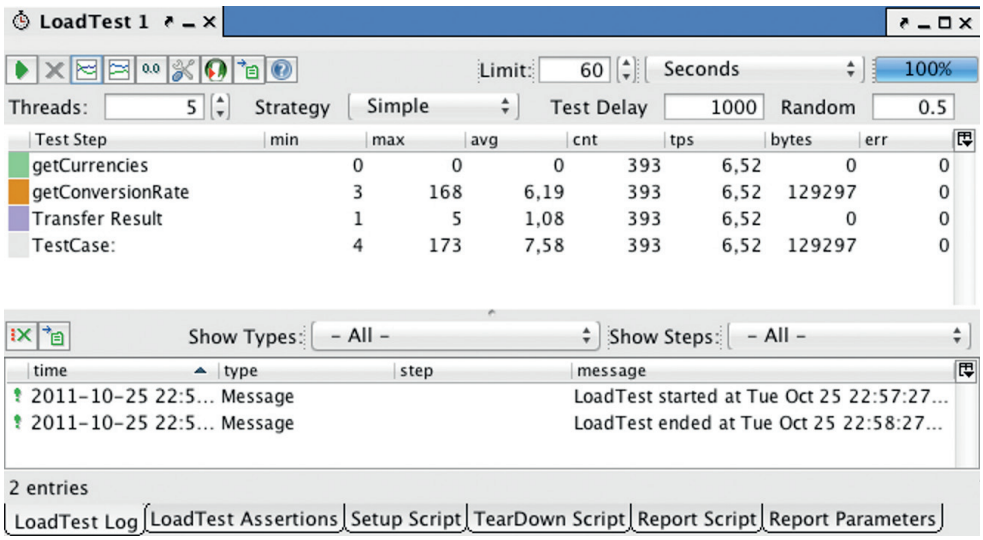


Abbildung 5: Konfiguration und Ergebnisse eines Lasttests

zurückgegeben wird, wenn die entsprechende Operation auf dem Mock-Service aufgerufen wird.

Will man die Antwort dynamisch gestalten, kann auf Properties oder Groovy-Skripts zurückgegriffen werden. Müssen die Mock-Services noch intelligenter ausfallen, kann über Groovy der Request analysiert und die Antwort abhängig von den Eingabedaten gemacht werden. Listing 2 zeigt dies für einen Mock-Service, der eine Suchabfrage simuliert. Der String, der vom Groovy-Skript zurückgegeben wird (wie „Item 1 Response"), entspricht dem Namen einer vordefinierten Antwort in soapUI. Diese wird vom Mock-Service an den Aufrufer zurückgeschickt.

Durch den Einsatz eines Groovy-Skripts lässt sich also ein dynamisches Dispatching realisieren. Dies erlaubt wiederum die Implementierung von sehr flexiblen Mock-Services. Diese lassen sich direkt in soapUI starten und stoppen. Über den definierten Endpoint können sie dann von extern angesprochen werden. Abbildung 4 zeigt das Konfigurationsfenster beim Anlegen eines Mock-Service. Alternativ lassen sich die Mock-Services mit wenigen Mausklicks als WAR exportieren und danach ohne weitere Anpassungen in einem eigenen Servlet-Container ausführen.

Security-Tests

Mit der aktuellen Version 4.0 von soapUI lassen sich neu auch Security-Tests durchführen. Diese bauen auf bestehenden funktionalen Testfällen auf und ergänzen diese, indem sie definierte Security-Scans durchführen. Damit können Services auf gängige Schwachstellen und Angriffsmöglichkeiten überprüft werden. Unter anderem lässt sich kontrollieren, ob ein Service anfällig ist für XML-Bomben, XPath- und SQL-Injection oder Cross-Site-Scripting. soapUI führt einen Testfall immer erst normal durch und wiederholt diesen dann für jeden angehängten Security-Scan. Fehlgeschlagene Tests werden nach der Durchführung angezeigt. Security-Testing ist ein umfassendes und komplexes Thema. Weiterführende Informationen sind auf der offiziellen soapUI-Homepage einsehbar, auf der auch Tutorials zu diesem Thema zu finden sind (www.soapui.org).

Lasttests

SoapUI unterstützt zudem das Ausführen bestehender Tests als Lasttests. Man wählt dafür einen Testfall aus und soapUI generiert daraus per Mausklick einen Lasttest. Für die Testdurchführung können verschiedene Parameter festgelegt werden: Die Menge (parallele Threads) und Art (fix, zufällig, definierte Spitzen) der Last sowie die Abbruchbedingungen (Anzahl der Requests, Fehler, Dauer) sind einstellbar. Abbildung 5 zeigt die Übersicht eines durchgeführten Lasttests.

Somit lassen sich Lasttests in soapUI mit geringem Aufwand erstellen. Für größere und aufwändigere Lasttests ist jedoch loadUI zu empfehlen. Dies ist eine Open-Source-Software von eviware, die gut mit soapUI zusammenarbeitet. Wie der Name vermuten lässt, ist loadUI auf die Durchführung von Lasttests spezialisiert. Der Vorteil ist, dass man Lasttests basierend auf soapUI-Testfällen erstellen kann. loadUI bietet für Lasttests spezifischere Konfigurationsmöglichkeiten als soapUI und ist kom-

fortabler zu bedienen. Es ist hauptsächlich für die Generierung einer großen Last vorteilhaft. loadUI-Agenten können auf mehreren Computern installiert werden. Der „Master" verteilt die Tests auf alle konfigurierten Agenten, welche die Lasttests durchführen und die Ergebnisse zurücksenden.

Test-Automatisierung

Es ist von grundlegender Bedeutung, dass Tests regelmäßig ausgeführt werden. Somit empfiehlt es sich, die Test-Durchführung zu automatisieren. soapUI enthält Konsolenskripts für Windows und Linux, mit denen Tests via Kommandozeile ausgeführt werden können. Mit Kommandozeilen-Parametern lässt sich die Testdurchführung steuern. Damit kann man beeinflussen, welche Testsuiten beziehungsweise Testfälle durchgeführt werden, und man kann Properties setzen, den Endpoint der SOAP-Requests überschreiben und weitere Konfigurationen vornehmen. Falls gewünscht, lassen sich Reports im Format von JUnit generieren, die Auskunft über die Testresultate geben.

Im Menü von soapUI gibt es den Punkt „Launch TestRunner". Dieser öffnet ein Fenster, in dem sich das oben beschriebene Konsolen-Skript parametrisieren und aufrufen lässt. Startet man die Ausführung, zeigt soapUI im Log-Fenster den Kommandozeilen-Befehl inklusive sämtlicher Parameter an. Der Befehl kann kopiert und für spätere Testläufe gespeichert werden, sodass man nicht weiter auf das grafische Frontend angewiesen ist. Wenn die Ausführung dieses Konsolen-Skripts in ein Ant-Skript gepackt wird, lässt sich das soapUI-Projekt einfach in eine Continuous-Integration-Umgebung einbinden. Listing 3 zeigt einen Ausschnitt aus einem solchen Beispiel-Skript. Es existiert ebenfalls ein Maven-Plug-in für soapUI. Tutorials zur Benutzung sind auf der offiziellen soapUI-Homepage zu finden (www.soapui.org).

Funktionen von soapUI Pro

Die kostenpflichtige Version von soapUI ergänzt die Standard-Edition um einige Funktionen, die man nicht mehr missen möchte, sobald man sie kennengelernt hat. Erwähnenswert diesbezüglich sind die verschiedenen Möglichkeiten im Zusammenhang mit dem sogenannten „Data Driven Testing". Der mit der Pro-Version

```
def groovyUtils =
    new com.eviware.soapui.support.GroovyUtils( context )
def holder =
    groovyUtils.getXmlHolder( mockRequest.requestContent )

def sessionid = holder.getNodeValue( „//sessionid" )

if( !context.hasProperty( sessionid ))
  return „Invalid Session Id Fault"

def searchstring = holder.getNodeValue( „//searchstring" )

if (searchstring == „Item 1")
  return „Item 1 Response"
else if (searchstring == „Item 2")
  return „Item 2 Response"
else if (searchstring == „Item 3")
  return „Item 3 Response"
else
  return „Unknown Response"
```

Listing 2: Response Dispatching für einen Mock-Service

```
<target name="soapUI-tests">
  <exec executable="${testrunner.cmd}"
      failonerror="yes"
      failifexecutionfails="yes">

    <arg value="-e ${service.endpoint}" />
    <arg value="-rajf" />
    <arg path="${report.directory}" />
    <arg path="${soapUI.project.file}" />
  </exec>
</target>
```

Listing 3: Ausschnitt eines Ant-Skripts zur Ausführung von soapUI-Tests

erhältliche DataSource TestStep kann Daten aus verschiedenen Quellen lesen, die anschließend für die Testdurchführung verwendet werden können, beispielsweise als Properties in SOAP-Requests oder in Assertions zur Validierung. Die Daten können unter anderem aus Excel-Dateien, Datenbanken via JDBC oder einfachen Textdateien gelesen werden. Platziert man am Ende seines Tests einen sogenannten „Data Source Loop", springt dieser zurück zur DataSource und liest den nächsten Datensatz, beispielsweise die nächste Zeile aus einem Excel-Dokument. Der Test wird mit diesen Daten erneut durchlaufen.

Um neue Testdaten zu generieren, kommt der DataGen TestStep zum Einsatz. Im einfachsten Fall werden die Daten in eine interne Liste eingetragen. Für jeden Testlauf wird ein Element der Liste zurückgegeben, sequenziell oder zufällig. Wenn dies nicht ausreicht, können die Daten dynamisch mit einem Groovy-Skript generiert werden.

Ein weiterer Vorteil von soapUI Pro sind die verschiedenen Assistenten. Per Mausklick lassen sich alle Properties einsehen, auf die man Zugriff hat, und man kann diese direkt an der gewünschten Stelle einfügen lassen. Xpath-Ausdrücke – beispielsweise bei Assertions oder Mock-Services – lassen sich per Mausklick erstellen und müssen nicht mehr von Hand geschrieben werden.

Generell ist bei soapUI Pro der ganze Arbeitsbereich übersichtlicher dargestellt. In der Standard Edition lassen sich geöffnete Fenster an keiner Stelle andocken, sondern schweben quasi im Arbeitsbereich umher. Wenn mehrere Fenster geöffnet sind, liegen diese oftmals übereinander. Es ist schwierig, das gewünschte Fenster zu finden und dies stört bei der Arbeit.

In soapUI Pro ordnen sich die Fenster in Tabs an. Dies gestaltet den ganzen Arbeitsbereich sehr übersichtlich. Das klingt nicht nach einer großartigen Erweiterung, vereinfacht jedoch das tägliche Arbeiten.

Folgende Aspekte verdeutlichen ebenfalls die Vorteile der Pro-Version:

- SOAP-Requests und -Responses werden in einer Formularansicht aufbereitet und es muss nicht mit rohem XML gearbeitet werden.
- TestCase Coverage zeigt an, welche und wie viele der im Webservice definierten Operationen und Elemente durch Tests abgedeckt sind.
- Projekte können nicht wie bei der Standard Edition nur als einzelne XML-Dateien abgespeichert werden, sondern als sogenannte „Composite Projects". Damit wird ein Projekt in einer Ordnerstruktur angelegt. Dies geschieht nach folgender Logik: ein Ordner pro Testsuite und eine XML-Datei pro Testfall. Dadurch wird die Arbeit mit Versions-Kontrollsystemen wesentlich erleichtert beziehungsweise ein paralleles Arbeiten an einem Projekt überhaupt erst möglich.

Wenn regelmäßig mit soapUI gearbeitet wird, ist die Anschaffung der Pro-Version somit durchaus empfehlenswert.

Fazit

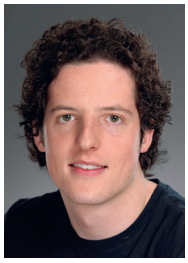
soapUI ist insgesamt als nützliche Bereicherung für das Software-Testing zu beurteilen. Man findet sich schnell darin zurecht und die Tutorials auf der offiziellen Homepage sind sehr gute Einstiegspunkte, um soapUI kennenzulernen. Alle Möglichkeiten, die soapUI bietet, werden dabei – zumindest grundlegend – abgedeckt.

Für Anliegen, die weiter in die Tiefe gehen, sind die Tutorials und Dokumentationen jedoch noch nicht ausreichend detailliert. Insbesondere die Dokumentation der soapUI-Java-API, die beim Schreiben von Groovy-Skripts häufig gebraucht wird, ist ziemlich dürftig. Im Internet sind weitere Antworten zu soapUI-Problemen verfügbar. In diesem Zusammenhang sind das offizielle Forum mit speziellen Boards für die Pro-Nutzer und folgende „Frage-und-Antwort-Seite" einen Besuch wert: www.stackoverflow.com.

Links

www.soapui.org
www.loadui.org
www.eviware.com
www.eviware.com/forum
www.smartbear.com
www.stackoverflow.com

Sebastian Steiner
sebastian.steiner@trivadis.com



Sebastian Steiner ist seit 2002 in der Informatik tätig und seit 2004 im Bereich der Software-Entwicklung mit Schwerpunkt Java. Von 2007 bis 2010 absolvierte er ein Informatikstudium an der Berner Fachhochschule. Seit Herbst 2010 arbeitet Sebastian Steiner für die Trivadis AG als Consultant im Java- und SOA-Umfeld.

Grails – die Suche ist vorbei

Stefan Glase und Christian Metzler, OPITZ CONSULTING GmbH

Grails ist ein Open-Source-Framework zur Entwicklung moderner Web-Applikationen auf Basis der dynamisch typisierten Programmiersprache Groovy und bewährten Technologien wie dem Spring-Framework, Hibernate und SiteMesh. Eine Vielzahl von Plug-ins macht es zudem möglich, wiederkehrende Problemstellungen mit bewährten Lösungen umzusetzen.

Grails baut auf der dynamischen Programmiersprache Groovy auf und profitiert auf diese Weise von einer extrem ausdrucksstarken und auch für einen Java-Entwickler in kürzester Zeit erlernbaren Sprache. Zugleich integriert sich Grails nahtlos in das Java-Ökosystem und kann auf bestehende Klassen und Bibliotheken zurückgreifen. Dies macht den Einstieg besonders dann interessant, wenn man aufgrund von Investitionen in die Java-Plattform an die Java Virtual Machine gebunden ist.

Convention over Configuration

Inspiriert von Ruby on Rails sind auch bei Grails die Paradigmen „Convention over Configuration" und „Don't repeat yourself" wichtige Erfolgsfaktoren für eine hohe Qualität und Produktivität bei der Entwicklung Grails-basierender Web-Anwendungen. So gibt Grails bereits in der Verzeichnisstruktur einen einheitlichen Rahmen vor, der es Entwicklern einfach macht, sich in neue Projekte einzuarbeiten.

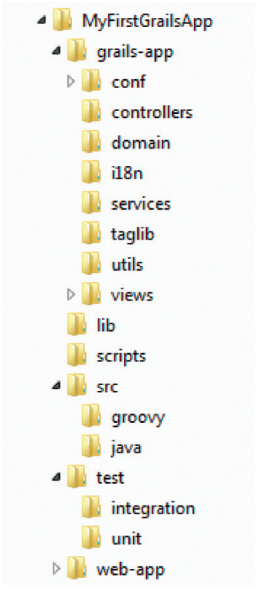


Abbildung 1: Verzeichnisstruktur eines Grails-Projekts

Bei Grails finden sich Artefakte eines bestimmten Typs immer am gleichen Ort innerhalb der Projektstruktur und folgen auch in der Namensgebung bestimmten Konventionen. So werden Konfigurationsskripte unterhalb des Verzeichnisses „grails-app/conf" verwaltet und lassen bereits durch ihren Namen die jeweils konfigurierten Aspekte erkennen:

- „ApplicationResource.groovy" bündelt zusammengehörende Ressourcen unter eindeutigen Namen zur einfacheren Einbindung
- „BootStrap.groovy" beschreibt die Ausführung von Code beim Starten und Stoppen der Applikation
- „BuildConfig.groovy" stellt alle erforderlichen Informationen zum Bauen der Anwendung bereit
- „Config.groovy" verwaltet die allgemeine Konfiguration der Anwendung wie beispielsweise das Logging
- „DataSource.groovy" definiert die zu verwendenden Datenquellen unter Berücksichtigung der Umgebung (Entwicklung, Test, Produktion etc.)
- „UrlMappings.groovy" beschreibt die Übersetzung von URLs in Aufrufe dafür vorgesehener Methoden

Modellierung von Fachklassen

Einen zentralen Bestandteil in Grails-Anwendungen stellen die „Domain Classes" dar, die unter „grails-app/domain" abgelegt werden und in der Regel fachliche Objekte der Anwendung repräsentieren. Unter der Haube greift Grails auf das bekannte Framework Hibernate zurück, um eine Abbildung dieser Domänen-Klassen auf relationale Datenbanken zu realisieren. Dieses objektrelationale Mapping wird durch Grails erweitert und nennt sich „Grails Object Relational Mapping" (GORM).

Mittels einer domänenspezifischen Sprache (DSL) können Domänenklassen um Validierungsregeln (Constraints) erweitert und Beziehungen zu anderen Klassen definiert werden. Auch die Abbildung in der Datenbank kann über eine DSL beispielsweise an bestehende Datenbank-Schemata oder unternehmenseigene Konventionen angepasst werden.

```
class Kunde {
    String name
    String email
    Date geburtsdatum
    Boolean geschaeftskunde
    String steuernummer

    static hasMany = [bestellungen: Bestellung]

    static constraints = {
        name(blank: false)
        email(unique: true, email: true)
        geburtsdatum(nullable: true)
        steuernummer(nullable: true)
    }

    String toString() {
        „$name ($email)"
    }
}
```

Listing 1

Listing 1 definiert die Struktur von Kunden-Objekten und bildet mittels des Schlüsselworts „hasMany" eine „1:n"-Beziehung zwischen Kunde und Bestellung ab. GORM erstellt dafür automatisch eine Zuordnungstabelle, um die Beziehung innerhalb der Datenbank abbilden zu können. Dank der Programmiersprache Groovy sind die ohne Sichtbarkeit angegebenen Properties der Klasse „Kunde" genauso wie die abgebildete Beziehung zu den Bestellungen des Kunden über implizite Getter und Setter sichtbar und veränderbar.

Validierung von Objekten

Des Weiteren werden Regeln (sogenannte „Constraints“) definiert, mit deren Hilfe die Domänenklasse validiert werden kann. So darf der Name des Kunden aufgrund des Constraints „blank: false“ nicht leer bleiben. Das Schlüsselwort „unique: true“ stellt die Eindeutigkeit der Werte sicher und bewirkt auch in der Datenbank ein entsprechendes Unique-Constraint für das Feld „email“, während „email: true“ lediglich applikationsseitig eine entsprechende Validierung auf eine gültige E-Mail-Adresse zur Folge hat. Dies ist aus Applikationssicht allerdings transparent.

Neben einer Vielzahl standardmäßig verfügbarer Validatoren können auch eigene definiert werden, um zum Beispiel Abhängigkeiten zwischen Eigenschaften der Klasse zu bestimmen. In Listing 2 machen wir die Steuernummer für Geschäftskunden obligatorisch.

```
steuernummer(validator: { val, obj ->
    if (obj.geschaeftskunde && !val) false
    else true
})
```

Listing 2

Repräsentation in der Datenbank

Das Mapping zur Datenbank – also die Abbildung auf Datenbanktabellen – kann ebenfalls mit einer einfachen Syntax (auch hier handelt es sich um eine DSL) beeinflusst werden. Im folgenden Beispiel (siehe Listing 3) wird der Tabellenname nicht den Konventionen überlassen, sondern explizit auf „customers“ geändert, und auch die Eigenschaft „geschaeftskunde“ wird in der Datenbank in einer Spalte mit dem Namen „businessCustomer“ gespeichert. Ferner soll die Beziehung des Kunden zu „bestellungen“ direkt beim Laden des Kunden aufgelöst und ebenfalls aus der Datenbank abgefragt werden.

```
static mapping = {
    table „customers“
    geschaeftskunde column: „businessCustomer“
    bestellungen lazy: false
}
```

Listing 3

Es besteht darüber hinaus noch eine Vielzahl weiterer Möglichkeiten, an dieser Stelle Einfluss auf die Datenbank zu nehmen, wie beispielsweise die Erzeugung von Indizes, das Caching-Verhalten, das Kaskadieren von Operationen über Beziehungen hinweg, Speichern des Datums der letzten Änderung und weitere bereits aus Hibernate bekannte Optionen.

Don’t repeat the DAO

GORM stellt zur Laufzeit weitere Funktionalität an den Domänenklassen bereit, um typische Interaktionsmuster mit der Datenbank nicht in jeder Applikation erneut implementieren zu müssen. Damit die ersten Daten in die Datenbank gelangen, müssen diese erst einmal gegen die in den Constraints formulierten Regeln validiert und im Erfolgsfall gespeichert werden (siehe Listing 4).

```
// Validieren und Speichern eines Kunden
def kunde = new Kunde(name: „Müller“)
if (kunde.validate()) {
    kunde.save()
} else {
    println kunde.errors
}
```

Listing 4

Sind die Daten nun in der Datenbank gespeichert, so möchte man sie zu einem späteren Zeitpunkt wieder im Zugriff haben. Hierfür bieten sich in den einfachen Fällen bei der Selektion der Daten die dynamischen Finder-Methoden an. Unsere Beispielklasse „Kunde“ bietet uns automatisch unter anderem folgende Zugriffsmethoden (siehe Listing 5).

```
// Erster Kunde mit dem Namen „Müller»
Kunde.findByName(‚Müller›)
// Erster Geschäftskunde mit dem Namen
„Müller»
Kunde.findByNameAndGeschaeftskunde(‚M
üller›, true)
// Alle Geschäftskunden
Kunde.findAllByGeschaeftskunde(true)
// Alle Kunden mit einem hinterlegten
Geburtstag
Kunde.findAllByGeburtsdatumIsNotNull()
```

Listing 5

Schnell wird deutlich, dass sich mit diesen Methoden bereits viele typische Fälle abdecken lassen, wenngleich es natürlich Situationen gibt, in denen dies nicht ausreicht. Wenn nach mehr als zwei Parametern gesucht werden soll, so existiert hierfür eine Funktion, der beliebige Suchparameter übergeben werden können (siehe Listing 6).

```
/* Alle Geschäftskunden mit dem Namen
„Müller», deren Geburtstag bekannt ist */
Kunde.findAllWhere(name: „Müller»,
        geschaeftskunde: true, geburtsdatum:
!null)
```

Listing 6

Komplexere Abfragen kann man mit der aus Hibernate bekannten Hibernate Criteria API formulieren. Hier gibt Grails dem Entwickler eine einfache, domänenspezifische Sprache an die Hand, mit deren Hilfe man gut lesbare Abfragen erzeugen kann (siehe Listing 7).

```
/* Alle Geschäftskunden, deren Name mit
„M“ beginnt, deren Geburtstag bekannt ist
und die mindestens eine Bestellung mit einem Warenwert größer als 5.000 Euro haben,
absteigend sortiert nach dem Geburtsdatum
*/
Kunde.createCriteria().list {
    like(„name“, „M%“)
    eq(“geschaeftskunde”, true)
    isNotNull(“geburtsdatum”)
    bestellungen {
        gt(“warenwert”, 5000)
    }
    order(“geburtsdatum”,“desc”)
}
```

Listing 7

Bei allen Abfragen, die potenziell mehr als einen Wert liefern können, besteht die Möglichkeit, zwei Parameter (Anzahl der Datensätze pro Seite und Offset des ersten zu selektierenden Datensatzes) für die seitenweise Aufteilung der Ergebnismenge (Paging) zu übergeben. Zusätzlich zu den dynamischen Finder-Methoden bietet Grails auch die Möglichkeit, nach einem Objekt zu suchen und, falls ein Objekt mit den Suchparametern nicht existiert, dieses zu erzeugen und zurückzugeben. Hier kann man wählen, ob dieses Objekt lediglich applikationsseitig erzeugt oder aber direkt in der Datenbank gespeichert werden soll.

Weiterhin erleichtert Grails das Arbeiten mit Beziehungen zwischen einzelnen Klassen. So existieren implizit Methoden, um Objekte zu einer „1:n“-Beziehung hinzuzufügen (siehe Listing 8).

```
// Hinzufügen einer neuen Bestellung zu
einem Kunden
Kunde.findOrSaveByName(„Müller»).
addToBestellungen(new Bestellung())
```

Listing 8

Standardmäßig arbeitet GORM mit dem Optimistic-Locking-Ansatz, es wird also zu jedem Objekt in der Datenbank eine Versionsnummer geschrieben, die bei Änderungen inkrementiert wird. Beim Speichern kann so die Versionsnummer im Objekt mit der in der Datenbank verglichen und gegebenenfalls auf die zwischenzeitliche Änderung des Objekts seit dem Laden reagiert werden.

Arbeiten mit Controllern

Grails implementiert das Model-View-Controller-Architekturmuster (MVC) und greift unter der Haube auf das Framework Spring Web MVC zurück. Controller bearbeiten Anfragen und erzeugen eine Antwort beziehungsweise bereiten diese vor. Ein Controller kann direkt eine Antwort generieren (beispielsweise für eine REST-Schnittstelle über XML oder JSON) oder er bereitet ein Modell auf und delegiert die Darstellung an eine View. Auch für Controller gelten Konventionen: Der Name eines Controllers muss mit dem Schlüsselwort „Controller“ enden und als Ablageort für diese Artefakte sind das Verzeichnis „grails-app/controllers“ oder ein Unterverzeichnis (im Sinne von Packages) vorgesehen (siehe Listing 9).

```
class KundeController {
    def list() {
        // Verarbeitung einer Anfrage
    }
}
```

Listing 9

Durch das standardmäßige URL-Mapping werden der Name des Controllers und die Namen der Actions, die innerhalb dieses Controllers definiert sind, in für Menschen lesbare Adressen übersetzt. Die oben dargestellte Action „list“ ist über den URI „/kunde/list“ (angehängt an die Adresse der Applikation) erreichbar. Demzufolge können wir in der Entwicklungsphase nach Bereitstellung auf dem Grails-eigenen Tomcat-Server unter „http://localhost:8080/myApp/kunde/list“ auf die Liste der Kunden zugreifen. Grails definiert grundsätzlich eine Standardaktion, die ausgeführt wird, wenn nur der URI des Controllers angegeben wird:

- Wenn nur eine Action existiert, ist das der Default
- Wenn eine Action mit dem Namen „index“ existiert, ist das der Default
- Alternativ kann der Default mittels „static defaultAction = „list““ gesetzt werden

Aus dem Controller kann aus jeder Action heraus auf verschiedene Informationen zugegriffen werden:

- „servletContext“ (auch bekannt als „Applikationskontext“): Daten über die gesamte Web-Applikation hinweg speichern
- „session“: Daten innerhalb einer Benutzersitzung ablegen
- „request“: Speicherung von Daten innerhalb der aktuellen Anfrage
- „params“: Zugriff auf eine Map mit den POST- und GET-Parametern der Anfrage
- „flash“: die Daten in diesem Scope sind nur im aktuellen und nächsten Request sichtbar

Der Flash Scope ist insbesondere für die Anzeige von Meldungen als Reaktion auf Aktionen des Anwenders interessant (siehe Listing 10). Die Nachricht, die in den Flash Scope geschrieben wurde, ist in der „list“-Action immer noch verfügbar und kann in der View entsprechend angezeigt werden.

```
def update() {
    def kunde = Kunde.get(params.id)
    if (!kunde) {
        flash.message = „Der Kunde mit der id
${params.id}
                wurde nicht gefunden.“
        redirect(action: list)
    }
    ...
}
```

Listing 10

Um aus dem Controller eine View zu erzeugen und dieser ein Modell zu übergeben, gibt es grundsätzlich mehrere Möglichkeiten. Entweder wird als Rückgabewert der Action explizit eine Map übergeben oder aber der Controller besitzt eigene Properties, die dann implizit zurückgegeben werden (siehe Listing 11).

```
// Rückgabe über explizite Map
def list() {
    [kunden: Kunde.list()]
}

// Rückgabe implizit über Properties des
Controllers
List kunden
def list() {
    kunden = Kunde.list()
}
```

Listing 11

In beiden Fällen wird Grails versuchen, die View darzustellen, die unter „grails-app/views/kunde/list.gsp“ liegt. Diese kann dann auf das Model zugreifen und das Ergebnis darstellen. Wo in Grails basierend auf dem Prinzip „Convention over Configuration“ auf Konventionen zurückgegriffen wird, besteht auch immer die Möglichkeit, mittels Konfiguration bei Bedarf von der Konvention abzuweichen. So wird im folgenden Beispiel nicht die standardmäßige View „list.gsp“ sondern eine „tabelle.gsp“ für die Anzeige verwendet (siehe Listing 12).

```
def list() {
    render(view: „tabelle“, model: [kunden:
Kunde.list()])
}
```

Listing 12

Rapid Prototyping mittels Scaffolding

In Zeiten der agilen Softwareentwicklung ist die Möglichkeit von Grails, in kürzester Zeit einen Prototyp zu erstellen, diesen sukzessive weiterzuentwickeln und an neue Anforderungen anzupassen, sehr wertvoll. An dieser Stelle unterstützt der Scaffolding-Mechanismus von Grails, mit dem auf der Grundlage der definierten Fachklassen entsprechende CRUD-Oberflächen (Create, Read, Update, Delete) bereitgestellt werden. Hierfür ist das Schlüsselwort „static scaffold = true“ im Controller erforderlich. Dies kann bereits bei der Generierung der Klasse automatisch eingefügt werden.

Die Grails-Konsole unterstützt mit einfachen Kommandos die Erzeugung von erforderlichen Artefakten und Teilen des Quellcodes. Abbildung 2 zeigt, wie mit dem Befehl „grails create-app auftragsverwaltung“ eine Grails-Applikation angelegt und der Applikationsrahmen geschaffen wird. In dieser Applikation werden mit „grails create-domain-class demo.Kunde“ die erste Fachklasse erzeugt und mit dem Befehl „grails create-scaffold-controller demo.Kunde“ die bereits vorgestellten CRUD-Funktionalitäten implementiert.

Die Applikation kann im Anschluss mit dem Grails-Kommando „grails run-app“ auf einer eigens dafür gestarteten Tomcat-Instanz bereitgestellt und ausprobiert werden. Durch das Scaffolding sind dabei nicht nur die Funktionalitäten im Controller bereitgestellt, sondern auch die dafür erforderlichen Views automatisch generiert.

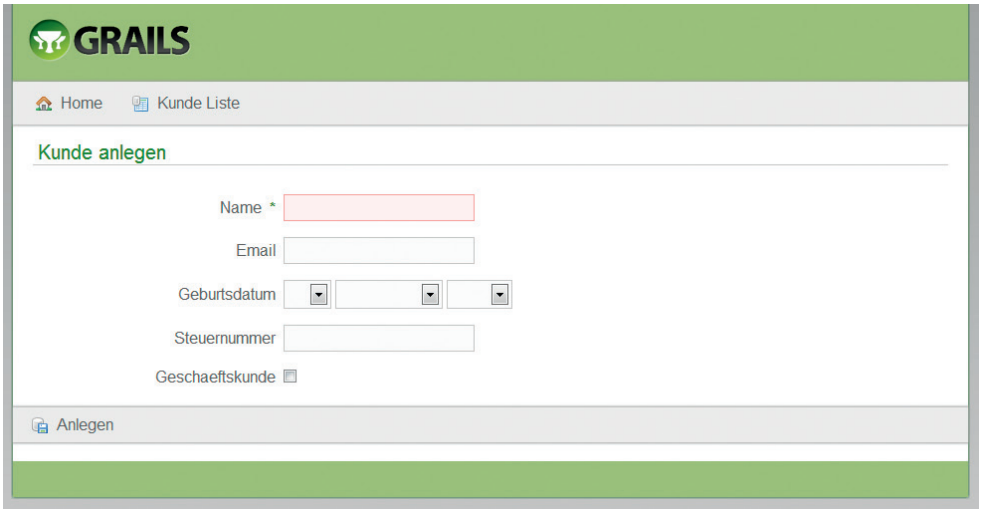


Abbildung 2: Erzeugung der Applikation mit der Grails Console

Abbildung 3: Listenansicht der existierenden Kunden

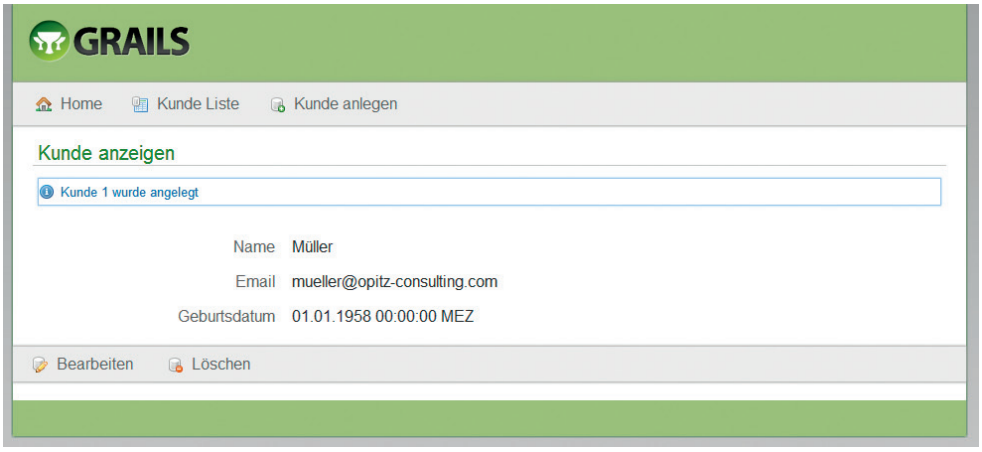


Abbildung 4: Maske zur Anlage eines neuen Kunden

Von einem solchen Stand ausgehend können nun Aktionen hinzugefügt, überschrieben und an die Anforderungen an die Applikation angepasst werden. In konkreten Projekten ist man so in der Lage, dem Kunden innerhalb weniger Tage (teilweise sogar weniger Stunden) einen ersten Eindruck von der Anwendung zu verschaffen und vor allem schnell festzustellen, ob man sein Problem verstanden hat und mit der Lösung auf dem richtigen Weg ist (siehe Abbildungen 3 bis 5).

Test Driven Development

Schon in der Abbildung 2 wird ersichtlich, dass automatisierte Tests einen hohen Stellenwert bei der Entwicklung mit Groovy und Grails einnehmen. Sämtliche Kommandos zur Erzeugung neuer Artefakte (Fachklassen, Controller etc.) legen automatisch auch die entsprechenden Unit-Test-Klassen an. Grails bietet für das Unit-Testing zusätzlich zu den bereits in Groovy von Haus aus verfügbaren Mock-Funktionalitäten ein eigenes Mock-Framework an, das uns erlaubt, die Funktionalität sämtlicher Artefakte einer Grails-Anwendungen unter Ausblendung der Abhängigkeiten zu Infrastruktur und anderen Artefakten durch Tests abzusichern.

Neben Unit-Tests können außerdem auch Integrationstests gegen eine laufende Datenbank automatisiert werden. Grails sieht hier wie aus Abbildung 1 ersichtlich schon bei der Erzeugung einer neuen Applikation das Verzeichnis „test/integration“ vor. Darüber hinaus existiert eine Vielzahl von Plug-ins, die zum Beispiel bei der Testdatengenerierung unterstützen oder funktionale Tests über die Oberfläche der laufenden Grails-Anwendungen ermöglichen.

Erweiterung der Anwendung durch Plug-ins

Grails liefert mit einem ausgefeilten Plug-in-Mechanismus die Antwort auf fehlende Funktionalität im Framework. Grails ist von Haus aus stark modularisiert aufgebaut und selbst einige der Basisfunktionen sind bereits in Plug-ins ausgelagert und können

so bei Bedarf hinzu- oder weggenommen werden. Grails-Plug-ins sind selber gültige Grails-Applikationen, die über einen Plug-in-Deskriptor genauer beschrieben werden.

Mittlerweile gibt es über 700 öffentlich verfügbare Plug-ins, die bei den verschiedensten Aufgaben unterstützen und mit dem Grails-Kommando „grails install-plugin <pluginname>“ zur eigenen Applikation hinzugefügt werden können. Zu den bekanntesten Plug-ins zählen:

- „hibernate-plugin“ stellt GORM als Brücke zwischen Hibernate und Grails bereit
- „mail-plugin“ liefert eine eigene domänenspezifische Sprache und die Funktionalität zum Versenden von E-Mails
- „quartz-plugin“ erlaubt die Steuerung von Jobs mittels Zeit-Intervall oder Cron-Expression innerhalb von Grails-Anwendungen
- „spring-security-core-plugin“ dient zur Absicherung der Anwendung unter Verwendung eines Rollen- und Rechtekonzepts für angemeldete Benutzer
- „file-uploader-plugin“ unterstützt beim Hochladen von Dateien unter Berücksichtigung von Dateigrößen, Dateitypen und Ablageorten
- „tomcat-plugin“ stellt einen Tomcat-Applikationsserver während der Entwicklung bereit, auf dem die Anwendung ausgeführt werden kann

Implementierung der Oberfläche

Die View-Logik in Grails basiert auf der Open-Source-Bibliothek „SiteMesh“. Diese ist ein Layout-Framework und hilft dabei, ein Template aufzubauen, das mit den Inhalten der verschiedenen Views dynamisch gefüllt wird. Dabei stehen insbesondere eine einheitliche Navigation und ein konsistentes „Look & Feel“ im Vordergrund.

Grails arbeitet mit Groovy Server Pages (kurz: GSPs). Diese bestehen aus HTML-Code, angereichert um GSP-Tags. Durch SiteMesh ist es möglich, einzelne Teile der GSP in Templates auszulagern. So gibt es in Grails standardmäßig eine „main.gsp“, die das Layout und auch die Importe für Style-Sheet-Definitionen enthält. Eine einfache GSP, um die Liste der Kunden anzuzeigen, könnte wie in Listing 13 aussehen.

```
auftragsverwaltung — java — 120×26
java
stefanglase:tmp stefanglase$ grails create-app auftragsverwaltung
| Created Grails Application at /private/tmp/auftragsverwaltung
stefanglase:tmp stefanglase$ cd auftragsverwaltung/
stefanglase:auftragsverwaltung stefanglase$ grails
| Enter a script name to run. Use TAB for completion:
grails> create-domain-class demo.Kunde
| Created file grails-app/domain/demo/Kunde.groovy
| Created file test/unit/demo/KundeTests.groovy
grails> create-domain-class demo.Bestellung
| Created file grails-app/domain/demo/Bestellung.groovy
| Created file test/unit/demo/BestellungTests.groovy
grails> create-scaffold-controller demo.Kunde
| Created file grails-app/controllers/demo/KundeController.groovy
| Created file grails-app/views/kunde
| Created file test/unit/demo/KundeControllerTests.groovy
grails> create-scaffold-controller demo.Bestellung
| Created file grails-app/controllers/demo/BestellungController.groovy
| Created file grails-app/views/bestellung
| Created file test/unit/demo/BestellungControllerTests.groovy
grails> run-app
| Server running. Browse to http://localhost:8080/auftragsverwaltung
| Application loaded in interactive mode. Type 'exit' to shutdown.
| Enter a script name to run. Use TAB for completion:
grails>
```

Abbildung 5: Detailansicht eines Kunden

```
<html>
  <body>
    <h1>Liste der Kunden</h1>
    <g:if test="${flash.message}">
      <div class="message"
role="status">${flash.message}</div>
    </g:if>
    <table>
      <g:each in="${kundeInstanceList}"
status="i" var="kundeInstance">
            <tr class="${(i % 2) == 0 ? 'even' :
'odd'}">
              <td>
          <g:link action="show"
id="${kundeInstance.id}">
            ${fieldValue(bean: kundeInstance,
field: "name")}
          </g:link>
        </td>
        <td>
          ${fieldValue(bean: kundeInstance,
field: "email")}
        </td>
        <td>
          <g:formatDate
date="${kundeInstance.geburtsdatum}"/>
        </td>
      </tr>
          </g:each>
    </table>
  </body>
</html>
```

Listing 13

Wiederverwendung von Oberflächenkomponenten

Sehr gut kann man im oben stehenden Codebeispiel den Gebrauch von Grails Taglibs sehen (beispielsweise „<g:formatDate/>“ zur Formatierung von Datumswerten). Diese sind in jeder GSP verfügbar und unterstützen durch die Wiederverwendung von Oberflächen-Komponenten bei der Vermeidung von redundantem Code.

Mit sehr geringem Aufwand ist es möglich, auch eigene Taglibs zu erstellen. Diese weiteren Artefakte werden unter „grails-app/taglib“ abgelegt und enden im Namen auf „TagLib“. Ein einfacher Tag kann wie in Listing 14 implementiert werden.

```
class KundenTagLib {
    static namespace = "kunden"
    def daten = { attrs, body ->
      def kunde = attrs.kunde
      out << "<div>"
      out << "<p>Name: ${kunde.name}</p>"
      out << "<p>Email: ${kunde.email}</p>"
      out << "</div>
    }
}
```

Listing 14

Eine auf diese Weise implementierte Taglib kann ab sofort in jeder View angesprochen und verwendet werden. Im Beispiel gibt „<kunden:daten kunde="${kundeInstance}" />“ den Namen und die E-Mail-Adresse des Kunden aus.

Fazit

Die vielversprechende Kombination der dynamischen Programmiersprache Groovy und der Web-Anwendungs-Plattform Grails, eine auf etablierten und erprobten Open-Source-Frameworks basierende Plattform, zeigt nicht nur in kleinen Beispiel-

Anwendungen ihre Stärken, sondern hat sich für uns auch bereits in großen Enterprise-Anwendungen erfolgreich unter Beweis gestellt. Durch eine ausdrucksstarke Sprache entsteht gut lesbarer und damit wartbarer Quellcode. Dank einer Vielzahl stabiler und funktionsreicher Plug-ins braucht so mancher Framework-Code nicht mehr selber geschrieben werden. Und aufgrund der strukturellen Konventionen innerhalb von Grails-Anwendungen fällt Entwicklern die Einarbeitung in neue Projekte um ein Vielfaches leichter. Alle diese Faktoren helfen dabei, dass die Fachlichkeit einer Anwendung wieder mehr in den Vordergrund rückt. Kunden werden es danken, wenn ihre Anforderungen in kürzerer Zeit umgesetzt und an geänderte Bedingungen angepasst werden können.

Stefan Glase
stefan.glase@opitz-consulting.com
Christian Metzler
christian.metzler@opitz-consulting.com

Stefan Glase (rechts) ist Senior Consultant im Bereich Application Engineering bei der OPITZ CONSULTING GmbH. Er beschäftigt sich seit mehreren Jahren mit der Architektur und Implementierung von Enterprise-Applikationen. Seine Schwerpunkt sind Enterprise Java mit dem Spring Framework sowie Groovy & Grails. Er ist Autor von Fachartikeln und spricht regelmäßig auf Fachkonferenzen.

Christian Metzler ist Consultant im Bereich Application Engineering bei der OPITZ CONSULTING GmbH. Er beschäftigt sich seit einigen Jahren insbesondere mit der Entwicklung von Web-Anwendungen mit Schwerpunkt auf der Sprache Groovy und dem leichtgewichtigen Framework Grails.

„Java besitzt immer noch ein enormes Potenzial …“

Usergroups bieten vielfältige Möglichkeiten zum Erfahrungsaustausch und zur Wissensvermittlung unter den Java-Entwicklern. Sie sind aber auch ein wichtiges Sprachrohr in der Community und gegenüber Oracle. Wolfgang Taschner, Chefredakteur von Java aktuell, sprach darüber mit Stefan Koospal, dem Vorsitzenden der Sun User Group Deutschland.

Wie bist du zur Sun User Group Deutschland gekommen?

Koospal: Ich bin 1993 zur Sun User Group Deutschland (SUG) gestoßen, weil ich als System-Administrator im Mathematischen Institut in Göttingen damals ein reines Sun-Umfeld zu verwalten hatte. Die jährlichen Treffen der User Group boten mir eine Möglichkeit, an wichtige Informationen zu gelangen und Kontakte zu anderen System-Administratoren zu knüpfen. 1993 war das Internet gerade erst dabei, groß zu werden, und die Verbindungen waren alle noch sehr schmalbrüstig. Da waren die CD-Kollektionen der SUG mit freier Software für Sun-Systeme eine wichtige Quelle, um mein System besser auszustatten.

Wie ist die Sun User Group Deutschland organisiert?

Koospal: Die SUG ist deutschlandweit als gemeinnütziger, eingetragener Verein organisiert.

Was zeichnet die Sun User Group Deutschland aus?

Koospal: Die SUG besteht jetzt fünfundzwanzig Jahre und kann damit schon auf etwas Geschichte zurückblicken. Wichtige Technologien wie das Internet und Java sind im Sun-Umfeld entstanden und gefördert worden. Anfang der 1990er Jahre bildeten Sun-Systeme das Rückgrat des Internets. IBM hatte ein eigenes Netz, Apple machte Appletalk, Microsoft sah im Internet keine wichtige Technologie und Linux war noch nicht so weit. Die Aufgaben des Vereins liegen inzwischen darin, die Nutzer von Sun-Technologien wie Java, Solaris, ZFS oder Staroffice weiter zu unterstützen, wenn nötig auch bei der Migration auf andere Systeme.

Wie viele Veranstaltungen gibt es pro Jahr?

Koospal: Wir veranstalten jährlich mit der Java User Group Deutschland e.V. die Source Talk Tage, bei denen wir Wissen zu verschiedenen Sun-Technologien wie Solaris, Staroffice und insbesondere Java an die Mitglieder der beiden Vereine und alle, die sich angesprochen fühlen, zu günstigen Konditionen weitergeben. Besonders wichtig für uns ist, dass unsere Mitglieder und Sponsoren den Studierenden die kostenfreie Teilnahme ermöglichen.

Was motiviert dich besonders, als Vorstand die Sun User Group Deutschland zu führen?

Koospal: Die Treffen der Sun User Group in den 1990er Jahren haben mir persönlich viel gebracht. Diese Aktivitäten waren bis 2005 ziemlich eingeschlafen, weil auch die Firma Sun nicht mehr an Anwendergruppen interessiert war. 2005 haben wir die Tradition mit den Source Talk Tagen wieder aufleben lassen und die Treffen inhaltlich um den Bereich „eLearning" erweitert. Dieser Kongress bietet inzwischen jedes Jahr fast 300 TeilnehmerInnen über drei Tage die Möglichkeit zum lockeren und fruchtbaren Erfahrungsaustausch und zur intensiven Wissensaneignung. Die Source Talk Tage ständig weiterzuentwickeln motiviert mich besonders stark.

Was bedeutet Java für dich?

Koospal: Das erste Mal habe ich 1994 bei einem Treffen der Sun User Group von Java gehört. Als dann klar war, dass Java wie vorgesehen auch wirklich von anderen Firmen intensiv genutzt wurde, hat sich die Java User Group Deutschland aus der SUG heraus gegründet. Ein Herstellerbindung im Namen passte nicht zur Java-Philosophie. Lange Jahre haben wir die Aktivitäten der JUG Deutschland e.V. mitfinanziert, weil wir das Konzept eines plattformübergreifenden, objektorientierten Programmiersystems und den Erfahrungsaustausch der Anwender unterstützen wollten.

Welchen Stellenwert besitzt die Java-Community für dich?

Koospal: Durch die Mitarbeit in der iJUG bietet sich mir die Möglichkeit, im Bereich Java unseren Vereinsauftrag, den ich oben skizziert habe, sehr effektiv umzusetzen.

Was hast du bei der Übernahme von Sun durch Oracle empfunden?

Zur Person: Stefan Koospal
Nach dem Mathematik-Studium in Göttingen mit Nebenfach BWL und Informatik begann er 1989 seine Tätigkeit als Software-Entwickler. Seit 1992 ist Stefan Koospal Systemverwalter am Mathematischen Institut der Universität Göttingen. Gleichzeitig ist er Vorstand der Sun User Group Deutschland.

Koospal: Ich dachte, jetzt gehe eine Ära zu Ende. Zuerst habe ich auch nicht verstanden, was Oracle mit Sun eigentlich will. Auf jeden Fall war mir klar, dass die Arbeit als Sun User Group neu ausgerichtet werden muss.

Wie sollte sich Java weiterentwickeln?

Koospal: Java besitzt immer noch ein enormes Potenzial. Besonders deutlich wird es für mich bei den Apps für Android. Klar, es ist kein vollständiges und lupenreines Java. Aber wie klein die Apps sind, wie schnell sie ablaufen und wie wenig Ressourcen sie benötigen, zeigt einige Vorteile von Java. Es wäre schade, wenn aus dem Streit zwischen Google und Oracle eine künstliche Abspaltung entstünde. Viel besser wäre es, wenn die getrennten Wege wieder zusammengeführt würden und auf der Android-Plattform ein komplettes Java verfügbar wäre.

Wie sollte Oracle deiner Meinung nach mit Java umgehen?

Koospal: Es ist klar, dass eine Firma Geld verdienen muss. Überzogene Rendite-Erwartungen an den Java-Bereich führen aber zum Niedergang des ganzen Projekts. Aus meiner Sicht war Java weder bei Sun, noch ist es jetzt bei Oracle richtig aufgehoben. Besser wäre es nach meiner Meinung, wenn sich eine Stiftung, in der sich alle ernsthaft interessierten Akteure zusammenfinden sollten, um die Weiterentwickelung von Java kümmern würde.

Wie sollte sich die Community gegenüber Oracle verhalten?

Koospal: Die Community sollte immer wieder auf allen Ebenen das Gespräch mit Oracle suchen und die Positionen klar machen. Verteufelung und Schmollen bringen nicht viel. Andererseits muss man sich dabei immer darüber im Klaren sein, dass Oracle eine sehr renditeorientierte Firma ist.

Kontakt:
koospal@uni-math.gwdg.de

Kapitän an Bord: Scrum als Match Race

Abbildung: ddp-images

von Uta Kapp, Allscout, und Jean Pierre Berchez, HLSC/Scrum-Events

Der Sport ist in manchen Bereichen hilfreich für die Entwicklung von Software. So können wir vom Segelsport etwas über den sinnvollen Einsatz von Personal und Bordcomputer für die Steuerung einer Yacht in einer Segelregatta lernen und für unsere tägliche Arbeit übernehmen.

Was wir vom America's Cup für die Softwareentwicklung lernen können: Eine Regatta hat feste Spielregeln. So wie der wohl bekannteste Segelwettbewerb, der America's Cup, in einzelnen Rennetappen (Match Races) stattfindet, so werden bei der agilen Software-Entwicklung fest definierte Zeitetappen von maximal dreißig Tagen gesetzt. Bei dem Projektmanagement-Framework „Scrum" heißen diese Etappen „Sprints".

Rollen

Eine klar definiere Rollenverteilung sorgt für die Organisation der Selbstorganisation. Eine der wichtigsten Rollen an Bord der Segelyachten ist die des Navigators. Dieser hat zwei simple und doch schwierige Aufgaben: Sicherzustellen, dass das Boot anhand der Gegebenheiten wie Wind, Strömung etc. schnell fährt und dass es in die richtige Richtung fährt. Er überprüft mithilfe seines Bordcomputers, ob das Geschwindigkeitspotential den Wetterbedingungen entsprechend ausgenutzt wird, und stellt sicher, dass Hindernissen aus dem Weg gegangen wird. Bei Scrum hat der „ScrumMaster" die Funktion eines Navigators. Er überwacht das Einhalten aller Prinzipien und hilft dabei, dass das Scrum-Team somit möglichst effektiv ans Ziel kommt.

Der Steuermann auf den Yachten führt das Ruder und gibt damit die Richtung an, in die das Schiff steuert. In Scrum entspricht dies der Rolle des „ProductOwner", der die Sicht des Kunden vertritt und bestimmt, was mit höchster Priorität umgesetzt wird.

Der Kapitän darf auf keinem Schiff fehlen. Er bestimmt, wohin die Reise geht, und hält die Vision aufrecht. Für jede Software-Unternehmung ist dies die Aufgabe des Chefs. Er steht hinter seinem Team und sorgt dafür, dass es ungehindert seine Arbeit verrichten kann. Im Gegenzug muss er, wie der Kapitän auf der Brücke, über den letzten Stand im Bilde sein. Das ist Transparenz. Das Team kann den Vorgesetzten und bei Bedarf weitere Stakeholder regelmäßig zu einem Executive-Meeting, dem „Sprint Review", einladen. Auch der ScrumMaster hat eine große Verantwortung und informiert das Management rechtzeitig über Hindernisse, die die Arbeit des Teams beeinträchtigen oder den Projektfortschritt verlangsamen. Hier sollten vor allem die Konsequenzen

dieser sogenannten „Impediments" und die dadurch erzeugten Kosten transparent gemacht werden.

Instrumente

So wie der Bordcomputer Informationen zur Position der Yacht in der Regatta liefert, so schaffen Scrum-Tools Transparenz bezüglich des Projektstands. Sie sind nützlich für die Verwaltung der zu erledigenden Aufgaben und der Planung. Das einfachste und sehr oft verwendete Tool ist hierbei eine Pinnwand mit Kärtchen und gegebenenfalls ein Excel-Sheet. Damit sollte jedes Team, das anfängt, sich mit Scrum zu organisieren, starten. So ist es möglich, zuerst im kleinen Kreis den Umgang mit den typischen Arbeitsabläufen zu erproben. Gemeinsam wird ein Taskboard gepflegt und bereits erledigte Aufgaben werden beispielsweise am Ende des Arbeitstages markiert. Teams und Projekte wachsen mit der Zeit und ein unübersichtlicher Dschungel an Aufgaben entsteht. Die Teams werden an verschiedene Standorte verteilt. Manche Teams arbeiten sogar über mehrere Zeitzonen. Der Einsatz von elektronischen Hilfsmitteln für die Verwaltung wird dann unerlässlich.

Selbstorganisation

Ein Kernkonzept in Scrum ist die Selbstorganisation des Entwickler-Teams. Dies ist ein radikal anmutendes Konzept im Vergleich zu klassischem Projektmanagement. Es gibt niemanden, der vergleicht, ob und wie schnell einzelne Teammitglieder Arbeitsschritte durchgeführt haben. Wie auf einer Segelyacht bringt jeder die Leistungen, die erforderlich sind, um die Etappe zu gewinnen. Da in einem multifunktionalen Team prinzipiell jeder an allen Aufgaben mitarbeiten kann, interessieren hier nur das gemeinsam geleistete Ergebnis und das, was noch gemeinsam zu erledigen ist. In einem sich selbst organisierenden Team ist jedes Mitglied selbstverantwortlich und proaktiv. Das ist ein sehr hoher Anspruch an alle Beteiligten. Kooperation, Kommunikation und Offenheit sind erforderlich. Jedes Teammitglied sollte zu jedem Zeitpunkt auf alle Informationen eines Projekts zugreifen können.

Scrum bezieht seine Effektivität aus der Fähigkeit der Beteiligten ihr Potenzial zusammenzuschließen, sodass das Ganze mehr wird als die Summe aller Teile. Sobald ein Konkurrenzverhalten der Teammitglieder untereinander gefördert wird, ist das Miteinander geschwächt. Metriken für die Verwaltung von Aufgaben sollten also immer nur das Gesamtsystem darstellen und nicht die geleistete Arbeit von Einzelpersonen. Der wirkliche Beitrag einer Einzelperson zum System ist nur in einer Fließbandproduktion in Arbeitseinheiten quantifizierbar, nicht aber in einem Entwicklungs- und Lernprozess.

Risiko-Management

Feste Zeitetappen sind das zentrale Element des agilen Vorgehens. Vor dem Start eines Sprints gibt es eine detaillierte Planung, das sogenannte „Sprint Planungsmeeting". Direkt im Anschluss findet eine Inspektion, die „Retrospektive" statt. Dieses Vorgehen schafft Kontrollpunkte, von denen auch das Management profitiert. Dem Team wird ein Vertrauensvorschuss von einem Sprint zugestanden. Während dieser Zeit erwartet das Team, dass es, ohne unterbrochen zu werden, in Selbstverantwortung das zuvor geplante Paket umsetzen kann. Für die Auftraggeber bedeutet dies ein maximales Risiko von einem Sprint, also von maximal dreißig Tagen.

Fazit

Ähnlich wie in einem Match Race fordert Scrum den ganzen Einsatz eines Software-Entwicklungsteams. Ein Team ist mehr als die Ansammlung von Personen. Wenn die Intelligenz, Kreativität und das menschliche Potenzial von Entwicklern zusammengeschlossen wird, dann ist das Ergebnis etwas völlig anderes als die Summe aller Teile. Scrum bietet uns ein Konzept, mit dem wir es schaffen, diese Ressourcen zu nutzen. Es hilft uns nicht nur, die logisch-rationalen Fähigkeiten zu nutzen, sondern auch unsere zutiefst menschlichen wie Intuition, Bauchgefühl, Empathie und Vertrauen. Mehr über das Scrum Framework steht im neuen Scrum-Guide der beiden Erfinder, Dr. Jeff Sutherland und Ken Schwaber, unter http://www.scrum-events.de/scrumguides/Scrum_Guide.pdf.

Uta Kapp
uta.kapp@scrum-events.de
Jean Pierre Berchez
jp.berchez@scrum-events.de

Uta Kapp ist freiberufliche IT-Beraterin und systemischer Coach. Mit einer Kombination aus Fach- und Prozessberatung für Softwareprojekte hilft sie Entwicklungsteams bei der Bewältigung der ständig steigenden Komplexität. Hier kommen agile Softwareentwicklungsmethoden wie Scrum und Kanban zum Einsatz. Uta Kapp ist zertifizierte Scrum Trainerin der Scrum.org und bietet regelmäßig Kurse für Einsteiger und Fortgeschrittene unter www.scrum-events.de an.

Jean Pierre Berchez ist ebenfalls zertifizierter Scrum Trainer der Scrum.org und beschäftigt sich seit mehr als 15 Jahren mit den Themen Projektmanagement und Software Engineering. Diese Themen vermittelt er als Lehrbeauftragter an den BA's Stuttgart und Heidenheim sowie an der Hochschule Lichtenstein. In den letzten Jahren liegt sein Interesse auf den Themengebieten agile Entwicklung mit Schwerpunkt Scrum sowie Application Life Cycle Management. Er organisiert unter anderem den Community-Event „Scrum-Day"(www.scrum-day.de) in Deutschland.

Rapid Java Power

Gerald Kammerer, freier Redakteur

XDEV 3 ist eine professionelle integrierte Entwicklungsumgebung zum Rapid Application Development für Java. Der Artikel zeigt nach Freigabe der finalen Version 3.0 den Entwicklungsstand und die neuen Features.

XDEV 3 ist eine vollständige Java-Entwicklungsumgebung wie Eclipse, NetBeans oder JDeveloper. Dem Entwickler stehen ein leistungsfähiger Java-Code-Editor mit Debugger und Java-Compiler zur Verfügung, sodass sich mit XDEV 3 grundsätzlich alles umsetzen lässt, was mit Java möglich ist. Anders als konventionelle Java-Editoren bietet XDEV 3 darüber hinaus ein umfangreiches Toolset für Rapid Application Development (RAD), dessen Einsatz keinerlei Java-Know-how voraussetzt und damit die Anwendungsentwicklung mit Java in vielen Punkten radikal vereinfacht, insbesondere die Erstellung grafischer Oberflächen, Datenbank-Zugriffe sowie das Databinding. Die gesamte Arbeitsweise erinnert gleich bei den ersten Schritten an Access, FoxPro und Oracle Forms Designer, sodass man von diesen Tools endlich schnell und einfach auf Java umsteigen und auf Anhieb beeindruckende Ergebnisse erzielen kann. Für Entwickler, die bereits über Java-Kenntnisse verfügen, sei an dieser Stelle erwähnt, dass der Einsatz von RAD-Features keinesfalls ein Muss ist. Bei jeder einzelnen Aufgabe kann man als Entwickler frei entscheiden, ob man eine der vorhandenen RAD-Funktionen unverändert einbindet, ob man diese bei Bedarf individuell erweitert oder lieber selber in Java implementieren möchte. Damit ist XDEV 3 die erste professionelle RAD-Entwicklungsumgebung für Java, die den Spagat zwischen „Drag&Drop"-Entwicklung und klassischer Java-Programmierung schafft und während der Entwicklung zu jeder Zeit einen fließenden Übergang ermöglicht. Nach über dreijähriger Entwicklung und einjähriger Betaphase ist nun die finale Version 3.0 wie angekündigt lizenzkostenfrei für Windows, Linux und Mac OS-X verfügbar (siehe www.xdev-software.de).

Swing-Oberflächen mit Drag&Drop designen

Viele Geschäftsanwendungen gelten gerade deshalb als nicht mehr zeitgemäß, weil ihre Oberflächen verstaubt wirken und sich nicht flexibel an die mittlerweile gängigen, hohen Bildschirmauflösungen anpassen können. Für die Modernisierung solcher Oberflächen ist Java die perfekte Technologie, denn Java Swing bietet alles, was

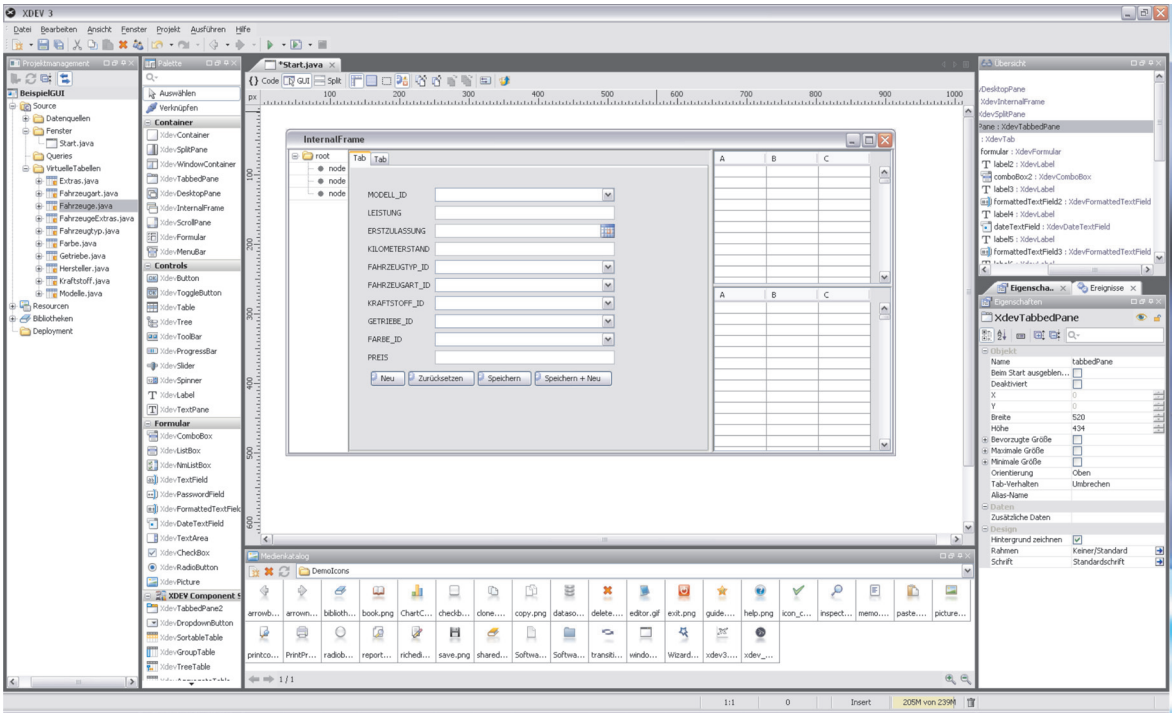


Abbildung 1: Die XDEV 3 Anwendung wirkt übersichtlich und ausgereift. Mit dem GUI-Builder lassen sich alle denkbaren Oberflächen sehr schnell und komfortabel umsetzen

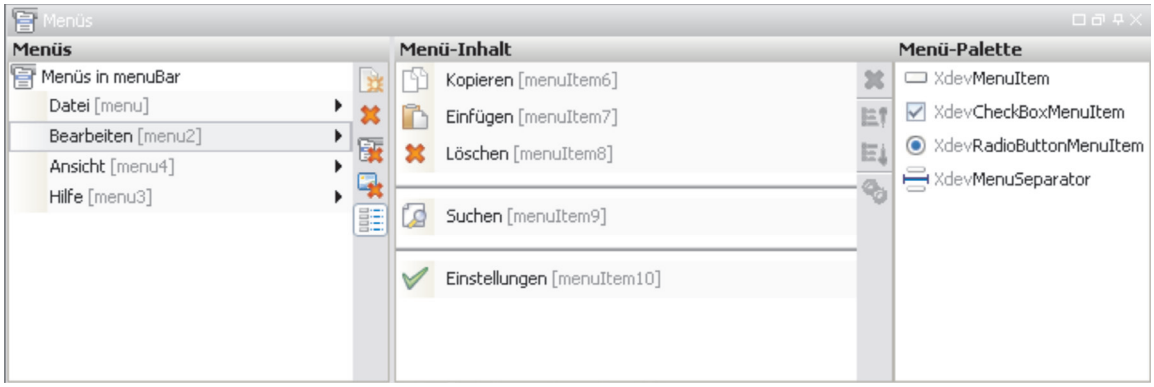


Abbildung 2: Menüleisten und Kontextmenüs werden mit dem neuen Menü-Assistenten in XDEV 3 per Drag&Drop einfach konstruiert

man für ein modernes Benutzer-Interface braucht, bereits „out of the box". Mithilfe von „Look&Feel" lässt sich zudem das Aussehen der kompletten Benutzeroberfläche mit wenigen Codezeilen vollständig ändern – ähnlich wie mit Windows-Themes.

Mit dem GUI-Builder, einem der Highlights von XDEV 3, lässt sich jede denkbare Benutzeroberfläche genauso schnell und einfach designen wie mit einem Grafikprogramm. Als Grafik-Bibliothek kommt Swing zum Einsatz. Da Swing enorm mächtig und über viele Jahre gereift ist, setzen die meisten Java-Entwickler es immer noch ein, während Abstract Window Toolkit (AWT) als veraltet gilt, das Standard Widget Toolkit (SWT) überwiegend im Eclipse-Umfeld verwendet wird und JavaFX sich in der Java-Community noch nicht nachhaltig genug durchgesetzt hat. Hinzu kommt, dass für Swing jede Menge „Look&Feels" und GUI-Komponenten von Drittanbietern frei oder für wenig Geld verfügbar sind. Eine zweite Variante für JavaFX hält der Hersteller jedoch für möglich, während ein GUI-Builder für AJAX-Applikationen auf Basis des Google-Web-Toolkit in den Labors der XDEV Software Corp. bereits als Prototyp existieren soll.

Mit XDEV 3 ist das Designen einer Oberfläche relativ einfach. In der Komponenten-Palette befinden sich alle Oberflächen-Elemente, die Swing für die Umsetzung grafischer Oberflächen zur Verfügung stellt, darunter Komponenten wie „Button", „Table", „Tree", „TabbedPane", „SplitPane", „Fenster" sowie alle wichtigen Formular-Komponenten. XDEV 3 verwendet jedoch von Swing abgeleitete Komponenten, da diese mit zusätzlichen Eigenschaften und Funktionen ausgestattet wurden. So sind sämtliche Controls bereits für die Umsetzung mehrsprachiger Oberflächen ausgelegt. Alle Komponenten lassen sich per „Drag&Drop" in die Arbeitsfläche einfügen, die dem späteren Programmfenster entspricht, und dort völlig frei und pixelgenau positionieren. Anders als bei Grafikprogrammen ist auch das Verschachteln von Komponenten möglich und bei vielen Komponenten sogar notwendig, insbesondere wenn diese andere Komponenten aufnehmen müssen wie die Menubar, Toolbar, TabbedPane (Registerreiter), Splitpane etc. Die dadurch entstehende Objekt-Hierarchie ist in einem separaten Fenster abgebildet.

Formulare generieren lassen

XDEV 3 ermöglicht nun auch das automatische Generieren voll funktionsfähiger Formulare. Alle dazu benötigten Informationen werden dabei von einer Tabelle abgeleitet, die man lediglich per „Drag&Drop" in den GUI-Builder einfügen muss. Zudem lässt sich jedes Formular jetzt auch mit einer Tabelle verknüpfen, sodass die in der Tabelle gelisteten Datensätze bei der Selektierung automatisch in das Formular übertragen werden. Neu ist, dass man jetzt jedes Formular sowohl für Eingaben als auch gleichzeitig als Suchformular verwenden kann. Das dazu notwendige Suchfeld fügt man einfach per „Drag&Drop" hinzu.

Nicht nur Formulare, sondern auch aufwändige Master-Detail-Filter-Ansichten sind nun schnell und ohne Programmierung konstruierbar. Dazu muss man lediglich zwei oder mehrere geeignete Komponenten wie Tabellen, List- oder ComboBoxen miteinander verknüpfen und anschließend per „Drag&Drop" eine virtuelle Tabelle zuweisen. Bei Master-Detail handelt es sich um die Abbildung einer „1:n"-Relation, beispielsweise zwischen Herstellern und Modellen. Bei der Auswahl eines Herstellers werden daraufhin automatisch alle Modelle dazu angezeigt.

Neu in Version 3.0 ist ein Menü-Assistent. Damit lassen sich neue Menüs per Mausklick anlegen, mit Menü-Items und mit Icons dekorieren. Per „Drag&Drop" fügt man seine Menüs dann entweder in eine Menubar am oberen Fensterrand ein oder direkt auf eine Komponente im Programmfenster, wo sich das Menü später zur Laufzeit als Kontextmenü aufrufen lässt. Wer mit Menü-Actions arbeiten möchte, kann diese wie in Swing üblich umsetzen.

Einfacher Umgang mit Layout-Manager

Fenster und Einzelkomponenten, die sich flexibel an die Bildschirmauflösung anpassen, werden in Java mithilfe des Layout-Managers umgesetzt. Am häufigsten fallen Border- und Gridbag-Layout an. Während beim Border-Layout lediglich das Andocken von Komponenten an die Seitenränder oder das Zentrieren einer Komponente möglich ist, erlaubt das Gridbag-Layout tabellarisches Layouten. Die Programmierung gilt jedoch als kompliziert und ist vor allem bei anspruchsvolleren Designs oft ein Geduldsspiel. In XDEV 3 steht dafür ein neuer Assistent zur Verfügung, mit dem sich selbst aufwändigste Tabellenlayouts schnell und einfach per Drag&Drop konstruieren lassen, beispielsweise mehrspaltige Formulare. Spätere Korrekturen können einfach durch Verschieben der Komponenten durchgeführt werden. Sogar das Verschieben ganzer Zeilen und Spalten via Cursor-Tasten ist jetzt möglich.

Fast in jedem größeren Projekt muss man GUI-Komponenten mit zusätzlichen Eigenschaften und Programmfunktionen ausstatten, die man jedoch nicht bei jeder Komponente neu programmieren möchte. Mithilfe der in Version 3.0 nochmals erweiterten GUI-Bean-Schnittstelle kann man jetzt von jeder GUI-Komponente ableiten, die XDEV 3 zur Verfügung stellt, auch direkt von den jeweiligen originalen Swing-Komponenten, diese beliebig erweitern und der Standard-Palette als neue GUI-Komponente, also als GUI-Bean hinzufügen. Auch das Einbinden externer GUI-Beans ist möglich. Damit lässt sich der GUI-Builder grenzenlos mit neuen Komponenten erweitern. Hinzu kommt das Instanziieren von GUI-Beans optional bereits zur Entwicklungszeit. Dies gewährleistet noch mehr WYSIWYG, ist jedoch nicht immer empfehlenswert. Falls eine Bean auf Ressourcen zugreifen möchte, die erst zur Laufzeit verfügbar sind, sollte man diese Funktion vorsorglich deaktivieren.

XDEV Component Suite

Java Swing stellt im Grunde nur einfache Standard-Komponenten zur Verfügung. Wer sich mehr Funktionsumfang wünscht, etwa eine Tree-Table (eine Kombination aus Table und Tree), muss tief in die Swing-Komponenten-Programmierung einsteigen, was ohne einschlägige Programmiererfahrung mit Swing in der Praxis nicht möglich ist. Mit der Component Suite steht dem Entwickler nun die umfangreichste GUI-Komponenten-Sammlung zur Verfügung, die für Java existiert. Die Highlights der Suite sind ein Text-Editor, mehrere Datagrids wie Tree-, Group- und Aggregate-Table, ein Fenster-Docking- Framework, mit dem Anwender ihre Oberfläche individuell organisieren können, sowie eine voll funktionsfähige Kalender-Komponente. Dazu bietet die Suite Funktionen wie Autovervollständigung, Quickfinder und mithilfe einer GUI-Persistence-API die Möglichkeit, sämtliche Nutzereinstellungen bezüglich einer Oberfläche zu persistieren. Mit solchen Features lässt sich jede Geschäftsanwendung kräftig aufwerten, ohne dass man dafür eine einzige Zeile Java-Code schreiben muss. Die Component Suite ist als Plug-in zusätzlich zu installieren.

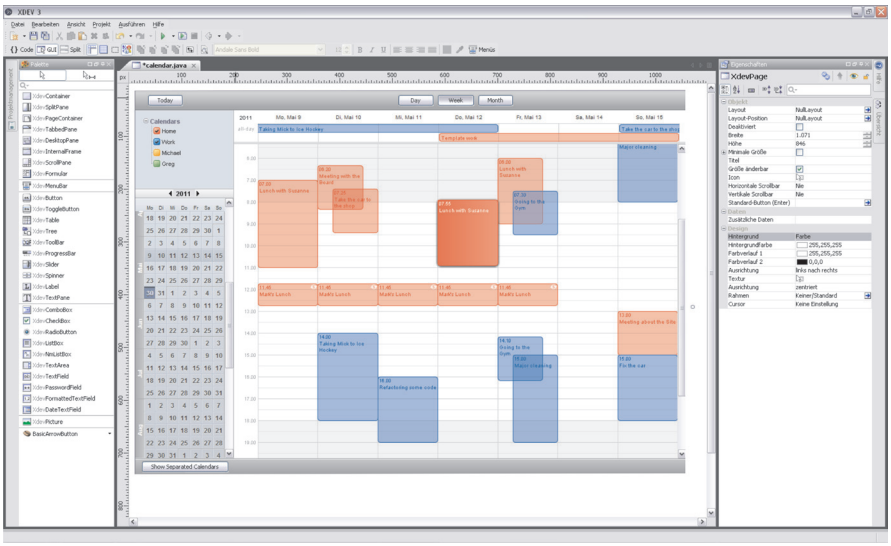


Abbildung 3: Java Swing bietet bereits eine Vielzahl an GUI-Komponenten, die XDEV 3 Component Suite erweitert diese um weitere hochwertige Profi-Komponenten (Screenshot: der voll funktionsfähige Kalender)

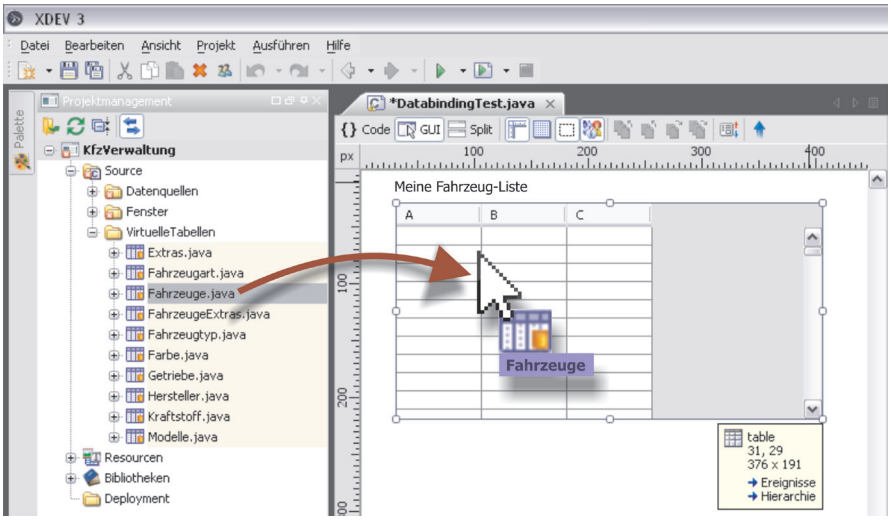


Abbildung 4: Einfaches Zuweisen der virtuellen Tabelle auf eine Table per Drag&Drop

Auch mehrsprachige Oberflächen lassen sich sehr einfach realisieren. Dazu legt man für jede Sprache, die man unterstützen möchte, eine Ressourcen-Datei an. Dabei handelt es sich um eine ganz gewöhnliche Textdatei, in der für jeden Text, der auf der Oberfläche erscheinen soll, eine Variable definiert wird, zum Beispiel „save = Speichern“. Auf der Oberfläche selbst werden dann nur noch die jeweiligen Variablen verwendet, die in geschweifte Klammern zu setzen sind und mit einem „$“-Zeichen beginnen, zum Beispiel „{$save}“. Zur Laufzeit ermittelt die Applikation dann automatisch über die Systemeinstellungen die festgelegte Sprache, greift auf die entsprechende Ressourcen-Datei zu und mappt sämtliche Texte mithilfe der definierten Variablen auf die Oberfläche. Per Methoden-Aufruf lässt sich die Sprache jederzeit auch programmatisch setzen.

Bekanntlich können auch andere Java-IDEs wie NetBeans mit einem GUI-Builder aufwarten und auch für Eclipse gibt es Alternativen. Das Entscheidende ist jedoch, dass sich damit nur die Controls auf den Bildschirm bringen lassen. Die gesamte Funktionalität einer Oberfläche sowie das Databinding muss der Entwickler größtenteils selber implementieren. XDEV 3 bietet dafür nun eine komfortable Lösung.

Das von XDEV 3 verwendete XDEV Application Framework stellt mit „virtuelle Tabellen“ (VT) ein Databinding zwischen

GUI und Datenschicht zur Verfügung, das Abfragen, Visualisierung und Persistierung weitestgehend automatisiert. Damit muss sich der Entwickler weder mit JDBC, Resultset-Verarbeitung und Transaktionsmanagement noch mit komplizierter MVC-Programmierung auseinandersetzen, was für Einsteiger eine enorme Vereinfachung darstellt. Das Application Framework ist Open Source und steht seit Version 3.0 unter der LGPL-Lizenz.

Automatisierte Datenbankabfragen

Um Datensätze auszugeben, muss man in XDEV 3 jetzt nur noch eine virtuelle Tabelle auf die Oberfläche ziehen, beispielsweise auf eine Table oder eine Listbox. Die Datenbank-Abfrage wird dann automatisch im Hintergrund erzeugt. Eine virtuelle Tabelle ist quasi eine leere Kopie einer Datenbank-Tabelle. Durch diese Vereinfachung wird der Entwickler vollständig vom Umgang mit JDBC-Resultset-Objekten und komplizierter MVC-Programmierung befreit (Model-View-Controller), was vor allem für Java-Einsteiger eine radikale Vereinfachung darstellt.

Für komplexere Abfragen bietet XDEV 3 einen komfortablen Query-Assistenten, mit dem man sich Joins sowie sämtliche Abfragebedingungen mit der Maus zusammenklicken kann. Auch alle wichtigen Aggregatfunktionen sowie Gruppierung, Sortierung und Limits werden jetzt zufriedenstellend unterstützt. Sehr innovativ ist die Abbildung logisch verknüpfter Bedingungen, die untereinander dargestellt werden. Anstatt Klammern zu setzen, werden die Terme eingerückt, wodurch auch bei zahlreichen Bedingungen die Übersicht erhalten bleibt. Als Parameter können im Assistenten auch Variablen und Methodenaufrufe angegeben werden. Der generierte Code besteht aus Java-Objekten, die sich sowohl mit dem Assistenten als auch direkt im Editor bearbeiten lassen. Erst zur Laufzeit werden die Statements in SQL umgewandelt.

Wer seine Abfragen lieber direkt in SQL formulieren möchte, kann auch SQL-Strings direkt absetzen oder dynamisch generieren lassen. Dafür stellt die API in der Klasse „DBUtils“ die Methode „query()“ zur Verfügung. Das Abfrage-Ergebnis wird damit wahlweise als Resultset-Objekt oder virtuelle Tabelle zurückgegeben.

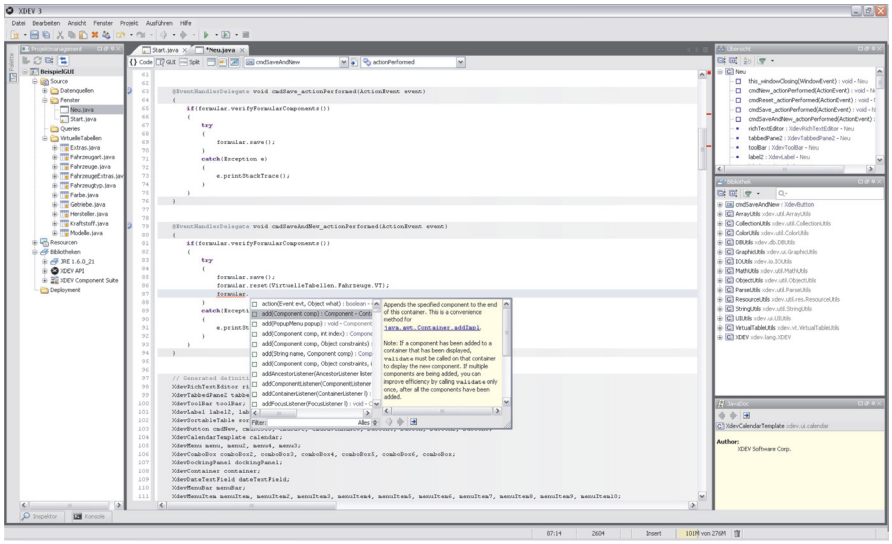


Abbildung 5: Unter der RAD-Haube ist XDEV 3 eine reine Java-IDE (analog Eclipse), die auch von Entwicklungsprofis aufgrund ihrer Funktionsvielfalt sehr geschätzt wird

Deployment

Wer mit XDEV 3 eine lauffähige Anwendung erzeugen möchte, muss sich nicht mit dem Schreiben von Build-Skripten auseinandersetzen, denn diese werden vollständig von der IDE auf Basis von „Apache Ant“ generiert. Grundsätzlich hat der Entwickler die Wahl zwischen Java-Applikation, Web-Anwendung und Webstart-Applikation. Die mitgelieferten Java-Datenbanken „HSQLDB“ und „H2“ kann man nun auch beim Deployen per Mausklick an seine Anwendung anbinden.

Als nächster großer Milestone steht eine SVN-Integration auf der Roadmap. Die Nutzung von Apache Subversion für das Entwickeln im Team sowie als Versionsverwaltung ist jedoch bereits jetzt schon mithilfe von Tools wie TortoiseSVN problemlos möglich. Eine direkte Integration verspricht jedoch weniger Arbeitsschritte und somit zusätzlichen Komfort.

Wie bereits erwähnt, soll es in naher Zukunft auch einen zweiten GUI-Builder für die Entwicklung von AJAX-Applikationen auf Basis des Google-Web-Toolkit (GWT) geben. Der Vorteil von AJAX-Frontends ist die Lauffähigkeit im Web-Browser ohne Java Runtime und damit auch die Ausführbarkeit auf beliebigen mobilen Geräten.

Fazit

XDEV 3 ist die erste professionelle Java-IDE für Rapid Application Development. Viele RAD-Funktionen und die Arbeitsweise damit erinnern sehr an Oracle Forms Designer sowie teilweise auch an Access und Visual Basic. Damit vereinfacht XDEV 3 die Entwicklung datengetriebener Business-Anwendungen in Java, garantiert kürzere Entwicklungszeiten im Vergleich zu jeder anderen Java-IDE und ermöglicht Forms- und anderen 4GL-Entwicklern von heute auf morgen, auf Java umzusteigen. Mit der Component Suite steht dem Entwickler die umfangreichste GUI-Komponenten-Sammlung zur Verfügung, die für Java Swing existiert. An der Basis ist XDEV 3 dennoch eine echte Java-IDE, die in Hinblick auf Funktionalität und Performance durchaus mit NetBeans und Eclipse mithalten kann. Für IT- und Fachabteilungen, die schnell und kostengünstig auf Java umsteigen und bereits nach kurzer Einarbeitungszeit umfangreiche Unternehmensanwendungen in Java umsetzen möchten, ist XDEV 3 ein gutes Werkzeug.

Gerald Kammerer
info@redaktion-kammerer.de

Gerald Kammerer ist als freier Redakteur seit 2006 für verschiedene Computerfachzeitschriften mit Schwerpunkt Java- und AJAX-Entwicklung tätig.

Microsoft und Java

Klaus Rohe, Microsoft Deutschland GmbH

Microsoft und die Java-Welt werden häufig als Gegensätze gesehen. Dabei hat Microsoft eine Reihe von interessanten, teilweise frei verfügbaren Angeboten für Software-Entwickler, die Java als Applikations-Plattform nutzen. Der Artikel stellt diese vor, verweist auf Web-Seiten, auf denen weitere detaillierte Informationen zu finden sind, und zeigt die Web-Adressen zum Download.

Der Microsoft SQL Server ist ein modernes und leistungsfähiges relationales Datenbank-Management-System. Aktuelle Version ist der Microsoft SQL Server 2008 R2. Es gibt verschiedene Editionen, die sich vor allem hinsichtlich ihrer Skalierungsmöglichkeiten und der Unterstützung von BI-Funktionalitäten unterscheiden. SQL Server 2008 R2 Express ist kostenfrei nutzbar. Nachfolgend sind einige Punkte aufgezeigt, die man im Kontext von Java/JEE beachten muss, außerdem wird die für Einsteiger in die SQL-Server-Entwicklung besonders interessante Express Edition vorgestellt.

Die SQL Server Express Edition gibt es in drei Varianten. Für die Einarbeitung ist die Version zu empfehlen, die in Tabelle 1 aufgeführt ist, da man hier das relationale Datenbanksystem mit dem benutzerfreundlichen grafischen Verwaltungswerkzeug plus Reporting-Funktionalität und integrierter Freitextsuche bekommt.

SQL Server 2008 R2 Express with Advanced Services	Benutzerfreundliche Version des SQL Server Express mit dem grafischen Verwaltungstool SQL Server Management Studio Express, Berichtsfunktionen und einer erweiterten, textbasierten Suche. Download: http://www.microsoft.com/downloads/de-de/details.aspx?displaylang=de&FamilyID=e08766ce-fc9d-448f-9e98-fe84ad61f135

Tabelle 1: Eine der drei Express-Editionen

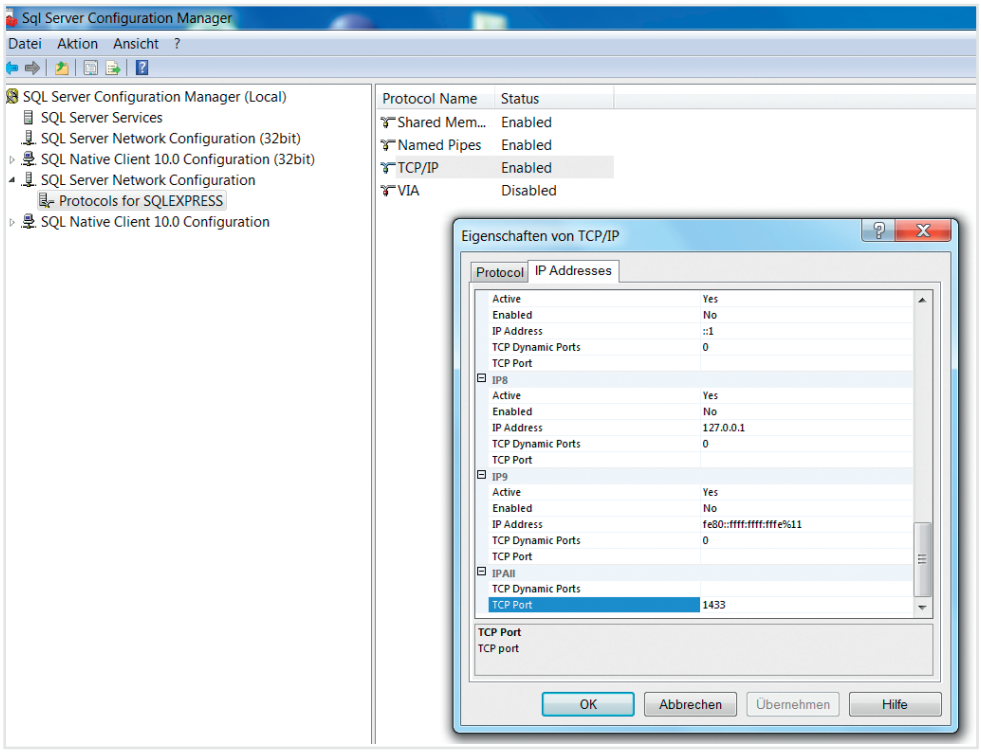


Abbildung 1: Konfiguration von SQL Server 2008 R2 Express für TCP/IP-Verbindungen im SQL Server Configuration Manager

Die Express Edition hat die folgenden Limitierungen:

- Nur eine CPU (Core)
- Maximal 1 GB Hauptspeicher
- Die maximale Größe einer Datenbank ist 10 GB. Eine SQL-Server-Instanz kann jedoch mehrere Datenbanken enthalten.

Um die Express Edition mit dem JDBC-Treiber zu nutzen, muss die Server-Instanz für TCP/IP-Verbindungen konfiguriert sein. Dies geschieht mit dem SQL Server-Konfigurationsmanager, der mit dem SQL Server-Management-Studio installiert wird. Man öffnet den Eintrag „SQL Server-Netzwerk-Konfiguration“ und klickt dann in der Liste auf den Eintrag „Protokolle für SQLEXPRESS“. In der rechten Hälfte des Fensters „SQL Server-Konfigurationsmanager“ muss dann der Wert für den Eintrag „TCP/IP“ auf „aktiviert“ gesetzt werden. Durch einen Doppelklick auf den Eintrag „TCP/IP“ wird das Fenster „Eigenschaften von TCP/IP“ mit zwei Tabs geöffnet. In „IP-Adressen“ geht man auf den untersten Eintrag „IP/ALL“. Hier sollte das Feld „Dynamische TCP-Ports“ leer und das Feld „TCP-Port“ auf den Wert 1433 gesetzt sein (siehe Abbildung 1). Port 1433 ist der Default-Port für TCP/IP-Verbindungen zum SQL-Server.

Damit die Konfigurationsänderung wirksam wird, muss der Server neu gestartet werden. Dies wird auch im SQL Server Configuration Manager durchgeführt, indem man den Eintrag „SQL Server-Dienste“ öffnet und dann im rechten Fenster mit einem rechten Mausklick auf den Eintrag „SQL Server (SQLEXPRESS)“ in dem sich öffnenden Fenster „Neu starten“ aktiviert. Anschließend kann man sich mit dem Kommando „netstat –a“ vergewissern, dass der Port 1433 auf dem lokalen Rechner abgehört wird.

Bei der Entwicklung von Applikationen auf der Basis von JEE-Applikationsservern kommen auch verteilte Transaktionen zum Einsatz, die auf der XA-Spezifikation aufbauen. Ein Beispiel für eine solche Transaktion ist die Änderung in einer Tabelle einer SQL-Server-Datenbank und das Schreiben einer Nachricht in eine JMS-Queue, wobei beide Operationen innerhalb einer Transaktionsklammer ausgeführt werden. SQL-Server unterstützt verteilte XA-Transaktionen, es ist allerdings sicherzustellen, dass der Windows-Dienst Distributed Transaction Coordinator (DTC) auf Windows gestartet wurde und für XA-Transaktionen konfiguriert wurde. Start und Konfiguration des DTC erfolgt über die Windows Management Console im Abschnitt „Dienste". Eine detaillierte Anleitung steht unter http://msdn.microsoft.com/de-de/library/aa342335.aspx.

Der SQL-Dialekt des Microsoft SQL Servers ist Transact SQL, kurz „T-SQL" genannt. Eine Einführung steht im T-SQL-Tutorium unter http://www.java2s.com/Tutorial/SQLServer/CatalogSQLServer.htm. T-SQL hat sehr große Ähnlichkeit mit Transact SQL des Sybase Datenbank-Servers, in dem der Microsoft SQL Server seinen Ursprung hat. Eine Historie dazu steht unter http://blogs.msdn.com/b/euanga/archive/2006/01/19/514479.aspx. Dort ist auch ein Verweis auf ein PDF-Dokument zu finden, das eine detaillierte Historie des SQL-Servers bis zur Version 2000 enthält.

JDBC-Treiber für den Microsoft SQL Server

Microsoft stellt mit dem „Microsoft SQL Server JDBC Driver 3.0" einen kostenfreien JDBC-Treiber für die SQL-Server-Versionen 2000 bis 2008 R2 und SQL Azure (siehe unten) zur Verfügung. Es handelt sich um einen „Type 4"-JDBC-Treiber, der kompatibel zur JDBC-4-Spezifikation ist. Den Treiber gibt es nicht nur für Microsoft Windows, sondern auch für Unix und Linux. Die Web-Seite zum Download ist http://www.microsoft.com/downloads/de-de/details.aspx?FamilyID=a737000d-68d0-4531-b65d-da0f2a735707#Instructions. Dort sind auch Installationshinweise für die unterstützten Betriebssysteme. Eine Dokumentation des JDBC-Treibers im HTML-Format ist nach der Installation im Verzeichnis „InstallationDirectory%\Microsoft SQL Server JDBC Driver 3.0\sqljdbc_3.0\enu\help" zu finden.

Abbildung 2 zeigt ein einfaches Java-Programm, das per JDBC auf die Datenbank „acronymdb" einer lokalen SQL-Server-Express-Instanz zugreift. Es nutzt die sogenannte „Integrated Security" des SQL-Servers. Bei dieser Art der Authentifizierung müssen keine Anmelde-Informationen angegeben werden, da die Identität und Berechtigungen des momentan im Windows-System angemeldeten Benutzers verwendet werden, unter dessen Kennung das Java-Programm ausgeführt wird, um sich gegen den SQL-Server zu authentifizieren. Dies ist im Connection-String durch „integratedSecurity=true" definiert worden. Damit dies funktioniert, muss in der Umgebungsvariable „PATH" der Pfad zum Verzeichnis der DLL „sqljdbc_auth.dll" eingetragen sein. Dies ist der Pfad zu einem der Unterverzeichnisse von „InstallationDirectory%\Microsoft SQL Server JDBC Driver 3.0\sqljdbc_3.0\enu\auth". Der Name des Unterverzeichnisses ist „x86", „x64" oder „IA64" – abhängig davon, ob man eine 32- oder 64-Bit-Version von Windows nutzt, und je nach der Intel-Prozessor-Architektur des benutzten Rechners. Hat man im Connection-String „integratedSecurity=false" gesetzt, so wird die SQL-Server-spezifische Authentifizierung eingesetzt und man muss den SQL-Server-Benutzer und dessen Passwort angeben.

Das gezeigte Programm ist standardmäßige JDBC-Programmierung. Spezifisch für den Microsoft SQL Server sind die Zeilen 13 und 14, in denen der Connection-String konstruiert wird, und die Zeile 15, wo der Microsoft-JDBC-Treiber geladen wird. Damit dieses Programm fehlerfrei kompiliert wird, muss die Umgebungsvariable „CLASSPATH" den Pfad „InstallationDirectory%\Microsoft SQL Server JDBC Driver 3.0\sqljdbc_3.0\enu\sqljdbc4.jar" enthalten.

Sollen mit dem Microsoft-JDBC-Treiber XA-Transaktionen genutzt werden, so

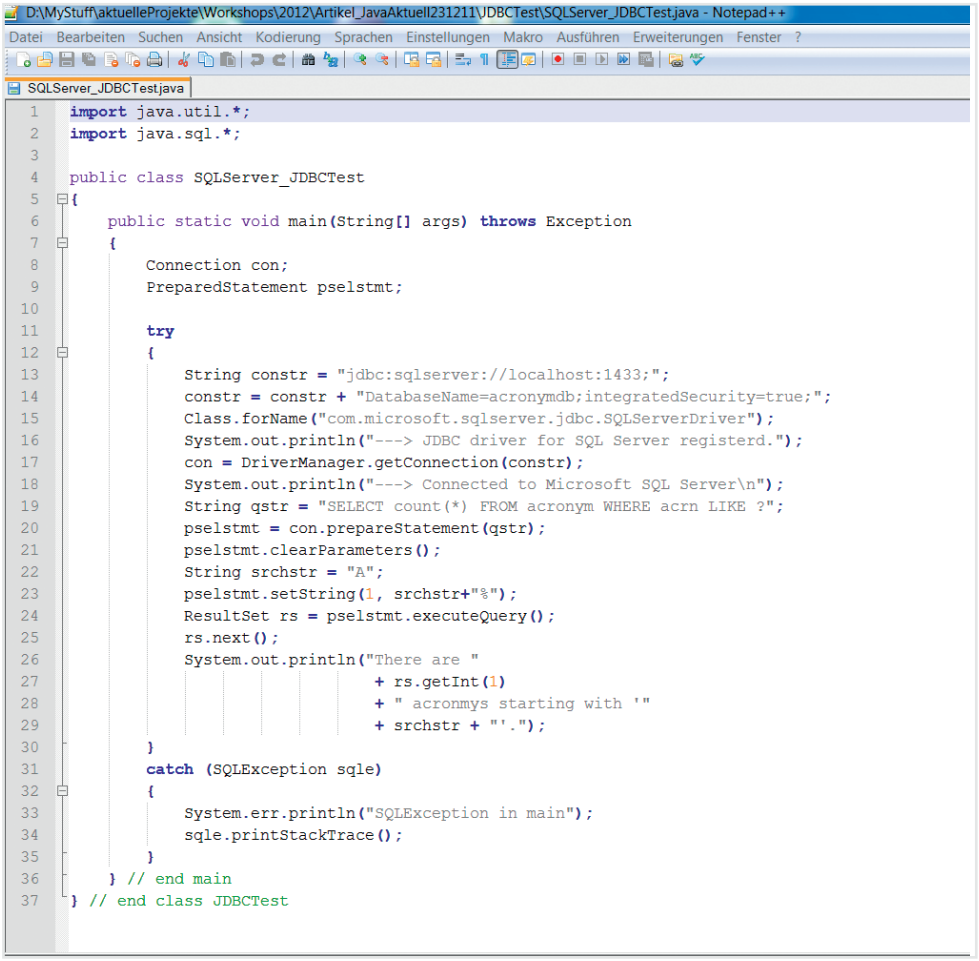


Abbildung 2: Einfaches JDBC-Beispielprogramm für den SQL-Server

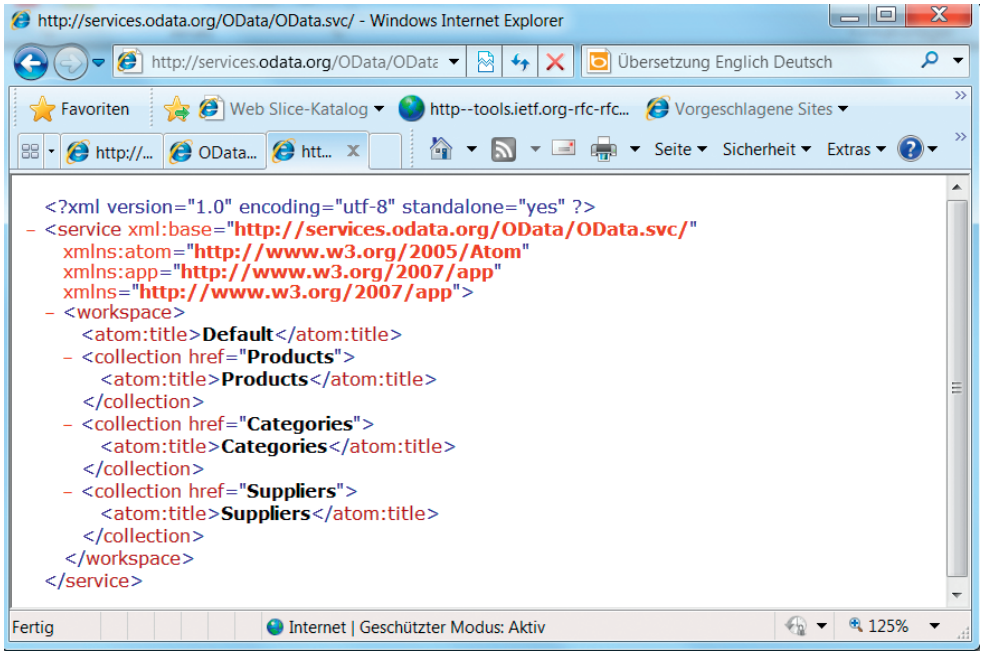


Abbildung 3: Ergebnis einer Browser-Abfrage des URI http://services.odata.org/OData/OData.svc/

muss der SQL-Server-Administrator das T-SQL-Skript „xa_install.sql" ausführen, das im Verzeichnis „InstallationDirectory%\Microsoft SQL Server JDBC Driver 3.0\sqljdbc_3.0\enu\xa" zu finden ist. Dabei sollte man die Anweisungen in den Kommentarzeilen am Anfang dieses Skriptes genau studieren.

Soweit ein Überblick zum aktuellen Microsoft-JDBC-Treiber. Im folgenden Abschnitt geht es auch um den Zugriff auf Daten, allerdings nicht beschränkt auf relationale Datenbanksysteme und nicht eingeschränkt auf eine spezifische Applikationsentwicklungsplattform.

Plattformübergreifende Daten-Integration mit dem Open Data Protocol

Das Open Data Protocol, kurz „OData" genannt, ist eine Spezifikation von Microsoft, um ein Protokoll für die plattformübergreifende Integration von Daten zu definieren. OData basiert auf REST und den Formaten „AtomPub" sowie „JSON". Außer für Microsoft .NET 3.5SP1 und 4.0 gibt es SDKs für Java, JavaScript, PHP und ObjectiveC. Im Folgenden werden OData und zwei Frameworks für Java vorgestellt. Die zentrale Web-Seite zu OData ist http://www.odata.org/.

Der Ursprung von OData sind die „WCF Data Services" in .NET 4.0. Im Oktober 2009 hat Microsoft die Spezifikationen des Protokolls und der Datenformate, welche die Basis der WCF-Data-Services bilden, allgemein zur Verfügung gestellt. Die Spezifikationen sind unter dem oben aufgeführten Link zu finden. Dort sind auch existierende OData-Consumer (Clients) und -Producer (Services) aufgeführt. Eine wichtige Rolle spielt OData auch im Zusammenhang mit Microsoft Windows Azure (siehe unten).

Jeder OData-Service veröffentlicht ein „Service Document" und optional ein „Service Metadata Document", das die Daten des Service mit einem abstrakten Datenmodell beschreibt, das als „Entity Data Model" (EDM) bezeichnet wird. EDM hat sehr große Ähnlichkeit mit dem Entity-Relationship-Modell von Chen (siehe http://de.wikipedia.org/wiki/ER-Modell). Das „Service Document" ist der Einstiegspunkt in einen OData-Service. Es liefert eine Liste aller Daten-Kategorien, die ein OData-Service bereitstellt, im AtomPub-Format. Der Einstiegspunkt des Service wird über einen Uniform Resource Identifier (URI) angesprochen wie http://services.odata.org/OData/OData.svc/. Dies ist der oben erwähnte Beispiel-Service, den man für Tests und eigene Experimente mit OData nutzen kann.

Spricht man diesen URI beispielsweise über den Web-Browser an, so bekommt man das in Abbildung 3 dargestellte XML-Dokument im AtomPub-Format zurück. Die Daten, auf die man über diesen Service zugreifen kann, sind also die „Collections", „Products", „Categories" und „Suppliers". Will man etwa auf alle Produkte lesend zugreifen, so muss ein http-GET-Request mit dem folgenden URI erzeugt werden: http://services.odata.org/OData/OData.svc/Products. Will man nur das Produkt mit dem Primärschlüssel 3 (ID in der Entität Products) ansprechen, so wird dies mit dem folgenden URI erreicht: http://services.odata.org/OData/OData.svc/Products(3).

Der Aufbau eines URI für OData ist exemplarisch in Abbildung 4 dargestellt. Die drei Komponenten eines OData-URI sind der „Service root URI", also der Einstiegspunkt in den Service, der „Resource path", der die Ressource spezifiziert, auf die eine Operation angewandt werden soll, und als dritte Komponente die „Query options", in der Filterbedingen etc. definiert werden können.

Weitere Details zur URI-Syntax sind unter http://www.odata.org/developers/protocols/uri-conventions zu finden. Da OData als Transportprotokoll HTTP/HTTPS und als Datenformate „AtomPub" oder „JSON" nutzt, können OData-Services im Prinzip von allen Applikationsplattformen, die Klassenbibliotheken für HTTP/HTTPS, XML und JSON zur Verfügung stellen, genutzt beziehungsweise auf diesen implementiert werden.

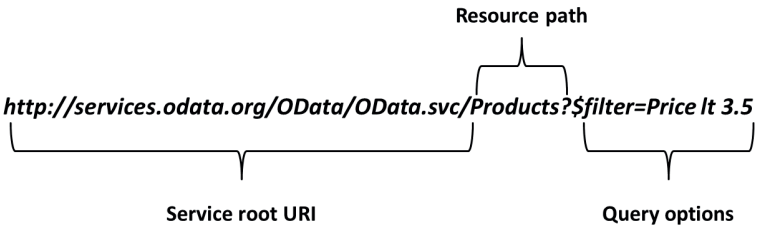


Abbildung 4: Komponenten eines URI für OData

Wesentlich vereinfacht wird die Implementierung von OData-Clients und -Services, wenn es dafür dedizierte Klassen-Bibliotheken gibt. Eine aktualisierte Liste der OData-SDKs findet man unter http://www.odata.org/developers/odata-sdk. Dort sind auch die Links zur Dokumentation und zum Download zu finden.

OData-SDKs für Java

Für Java gibt es zwei Klassen-Bibliotheken zu OData: „Restlet OData Extension" und „OData4j". Beide sind frei verfügbar und quelloffen. „Restlet OData Extension" ist eine Erweiterung des bekannten Restlet-Frameworks zur Vereinfachung der Implementierung von RESTful-Services in Java. Sie unterstützen in der aktuellen Version Restlet 2.1 nur die Implementierung von OData-Clients. Zu den OData-Extensions steht ein ausführliches Tutorium unter http://wiki.restlet.org/docs_2.1/13-restlet/28-restlet/287-restlet.html.

„OData4j" ist ein Framework, mit dem man sowohl OData-Clients als auch OData-Services implementieren kann. Des Weiteren ermöglicht OData4j auch die Programmierung von OData-Clients für Android. OData-Services können auch für die Google AppEngine entwickelt werden. OData4j basiert auf „Jersey" (siehe https://jersey.dev.java.net/) und nutzt „Joda Time" (siehe http://joda-time.sourceforge.net/). OData4j ist aktuell in der Version 0.5 verfügbar. Es kann als zip-Archiv von http://code.google.com/p/odata4j/downloads/detail?name=odata4j-archive-0.5.zip heruntergeladen werden. Entpackt man das zip-Archiv, so entsteht ein Verzeichnis mit dem Namen „odata4j-archive-x.y" (aktuell: x.y = 0.5) mit zwei Unterverzeichnissen „bundles" und „odata4j". Alle benötigten jar-Dateien sind in den Unterverzeichnissen enthalten. Die Dokumentation befindet sich im Unterverzeichnis „javadoc" von odata4j. Das Verzeichnis „bundles" enthält die jar-Files: odata4j-bundle-x.y.jar und odata4j-clientbundle-x.y.jar. Das Einfügen von „odata4j-bundle-x.y.jar" in „CLASSPATH" ist hinreichend für die Implementierung von Clients und Anbietern. Eine Reihe von Beispielen für fertig implementierte OData-Clients und -Services findet man unter http://code.google.com/p/odata4j/source/browse/odata4j-core/src/test/java/org/odata4j/examples/?name=0.5.

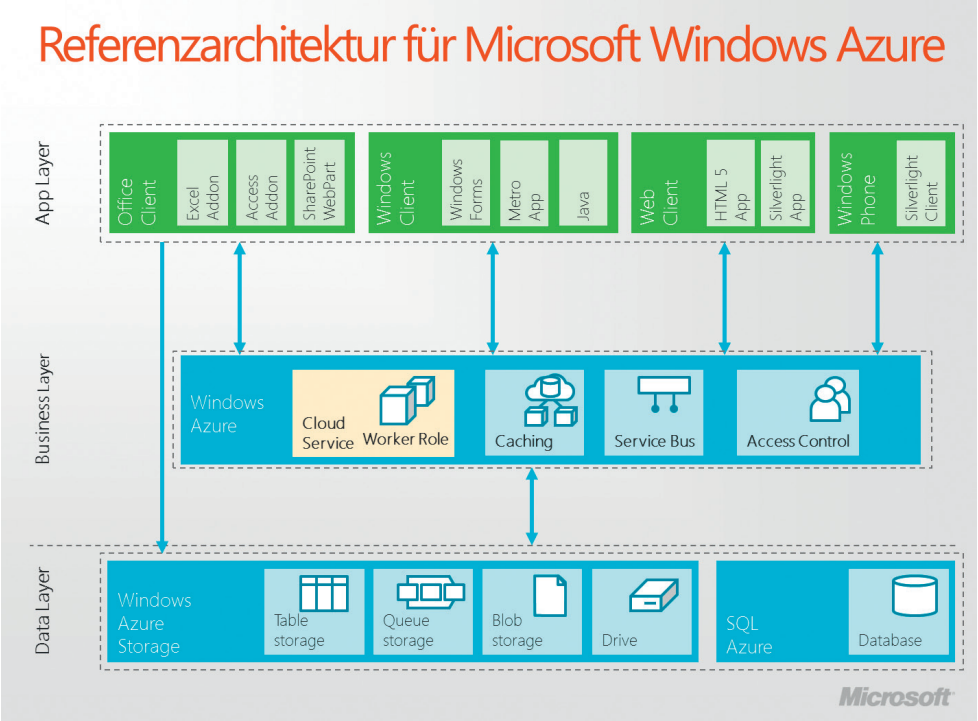


Abbildung 5: Referenz-Architektur für Windows Azure

Java-Entwicklung für Microsoft Windows Azure

Microsoft Windows Azure ist eine Cloud-Plattform in der Kategorie „Platform as a Service" (PaaS). Mit „Plattform" meint man hier Werkzeuge und Infrastruktur, die den Programmierer dabei unterstützen, Anwendungen zu entwickeln, die möglichst optimal für das Cloud Computing geeignet sind. Hier setzt das „Windows Azure SDK for Java" auf, das Eclipse als IDE nutzt. Es bietet Java-APIs für den Zugriff auf alle Komponenten, die in der Referenz-Architektur für Windows Azure in Abbildung 5 dargestellt sind. Die Web-Seite dazu ist http://www.windowsazure.com/en-us/develop/java/. Einen allgemeinen Überblick über das Azure-Programmiermodell findet man in http://www.davidchappell.com/writing/white_papers/The_Windows_Azure_Programing_Model_1.0--Chappell.pdf.

Beginnen wir mit der „Worker Role". Dies ist eine virtuelle Maschine, in der die Java-/JEE-Applikation ausgerollt wird. Dies kann eine einfache Web-Anwendung sein, die nur Servlets nutzt, oder eine komplexere JEE-Applikation, die beispielsweise noch EJBs nutzt und Web-Services über SOAP oder REST zur Verfügung stellt. Als relationale Datenbank müssen diese Applikationen SQL Azure über den oben besprochenen JDBC-Treiber nutzen, falls sie eine solche Datenbank benötigen.

Die Applikationen können aber auch den Azure-Table-Storage nutzen, der dem Konzept einer NoSQL-Datenbank entspricht, oder aber auch den Azure-Blob-Storage, in dem zum Beispiel Multimedia-Daten effizient gespeichert werden können. Eine weitere Speicherart ist das Azure Drive, das einem Plattenlaufwerk entspricht, auf dem man über das Java-IO-System Dateien speichern kann. Eine Azure-Java-Applikation kann auch über den Queue-Storage mit einer weiteren Worker Role asynchron kommunizieren, in der beispielsweise eine weitere Java-Applikation oder auch eine .NET-Applikation abläuft. Queue Storage wird allgemein dazu genutzt, CPU-intensive Aufgaben asynchron an spezielle Worker Roles auszulagern oder auch zur asynchronen Integration von Azure Cloud-Applikationen. Über den Azure Service Bus können Web-Services, die von lokalen Applikationen bereitgestellt werden, in die Cloud-Applikation integriert werden.

Azure Access Control ist eine Komponente, über die Authentifizierung und Autorisierung erfolgen kann. Sie unterstützt die Standards WS-Federation, WS-Trust,

OpenID 2.0 und OAuth 2.0. Azure Caching ist ein verteiltes Cache-System, auf das unterschiedliche Azure-Applikationen zugreifen können. Häufige Nutzungsszenarien sind das Speichern von Session-Zuständen oder Daten aus Datenbanken, die häufig gelesen werden.

Alle hier angesprochenen Azure-Komponenten sind über Java-APIs nutzbar. Die Management-Schnittstellen von Windows Azure sind als OData-Services ausgelegt, sodass sie von allen Programmiersprachen nutzbar sind, für die es ein OData-SDK gibt. Aber auch hierfür gibt es im Azure-Java-SDK schon Klassen, die diese Schnittstellen kapseln.

Ein wesentlicher Aspekt von Cloud Computing ist der, dass man Rechen- und Speicher-Ressourcen nur zu bezahlen braucht, wenn man sie nutzt, und dass sie dann auch sofort bereitstehen. Man muss daher nicht Rechen- und Speicherkapazität für Spitzenlasten anschaffen, die nur zu einem kleinen Teil der Nutzungszeit auftreten. Wenn man eine Cloud-Applikation entwickelt, muss man sich daher auch darüber im Klaren sein, was die Applikationen im Betrieb kosten. Microsoft bietet dafür auf http://www.windowsazure.com/en-us/pricing/calculator/ ein Werkzeug an, mit dem man verschiedene Optionen für eine Azure-Applikation durchspielen kann, um Werte für die Kosten zu erhalten, welche die Anwendung im Betrieb verursachen kann.

Ein weiterer Aspekt, den man bei Cloud-Applikationen beachten muss, ist der Datenschutz, wenn personenbezogene Daten verarbeitet werden. Weitere Informationen dazu gibt es für Windows Azure unter http://itcompliance.wordpress.com/.

Klaus Rohe
klrohe@microsoft.com

Klaus Rohe arbeitet bei der Microsoft Deutschland GmbH in der Developer Platform & Strategy Group und berät Software-Hersteller bei der Einführung und Anwendung von neuen Microsoft-Technologien aus dem Bereich Software-Entwicklung.

Kurz berichtet:

NetBeans IDE 7.1 steht zum Download bereit

Für die NetBeans-Plattform gibt es nun neue APIs für Windows-Systeme und eine Mehrfachfenster-Unterstützung. Auch das Verhalten vom TopComponentGroup wurde verbessert. Zusätzlich hat Oracle einen visuellen Layout-Designer für Fenster sowie steckbare Komponenten integriert, die Multi-Views möglich machen. Oracle verspricht mit NetBeans IDE 7.1 eine erhöhte Produktivität, die Unterstützung von Weblogic Server 12c sowie signifikante Verbesserungen für Swing GUI Builder, die CSS3-Unterstützung sowie Tools für ein grafisches Debuggen der Benutzeroberflächen JavaFX und Swing.

Die neue Version bietet eine JavaFX-2.0-Unterstützung aller drei Deployment-Modelle Desktop, Applet und JNLP. Sie ist auch mit einem Preloader ausgestattet, der den Anwendern eine verbesserte Ansicht des Ladeflusses einer Applikation garantieren soll und anhand von CSS3 UI-Steuerelemente anpasst.

Was Java anbelangt, erlauben nun die Refactoring Tools eine projektweite Verbesserung und Aktualisierung des Codes. Der neue visuelle Debugger unterstützt die Entwickler bei der Kontrolle vom Ausführungsfluss in verschiedenen Ebenen der grafischen Benutzeroberfläche. Eine Gap-Unterstützung im GridBagLayout ermöglicht, die Abstände zwischen Spalten und Zeilen aufrechtzuerhalten. Zudem können die Hervorhebung der Formatierung sowie die Funktion „inspect and refractor" ab jetzt bei allen selektierten Projekten innerhalb einer Applikation angewendet werden. Unterstützung für CSS3 gibt es nun in dem NetBeans CSS-Editor, der mit Code-Vervollständigung, Syntaxfärbung und Dokumentation der neuen CSS3-Elemente ausgestattet ist.

Zum Download: http://netbeans.org/downloads/index.html

Impressum

Herausgeber:
Interessenverbund der Java User Groups e.V. (iJUG)
Tempelhofer Weg 64, 12347 Berlin
Tel.: 0700 11 36 24 38
www.ijug.eu

Verlag:
DOAG Dienstleistungen GmbH
Fried Saacke, Geschäftsführer
info@doag-dienstleistungen.de

Chefredakteur (VisdP):
Wolfgang Taschner,
redaktion@ijug.eu

Chefin von Dienst (CvD):
Carmen Al-Youssef,
office@ijug.eu

Titel, Gestaltung und Satz:
Claudia Wagner
DOAG Dienstleistungen GmbH

Anzeigen:
CrossMarkeTeam, Ralf Rutkat,
Doris Budwill
redaktion@ijug.eu

Mediadaten und Preise:
http://www.ijug.eu/images/vorlagen/2011-ijug-mediadaten_java_aktuell.pdf

Druck:
adame Advertising and Media GmbH Berlin
www.adame.de

Java aktuell
Magazin der Java-Community

Apache Camel als Java Mediations-Framework im Vergleich zu kommerziellen Werkzeugen

Frank Erbsen, X-INTEGRATE Software & Consulting GmbH

Wie kann durch Mediation auf der Basis eines Messaging-Systems lose Kopplung zwischen Anwendungen erzielt werden? Gibt es bei der Umsetzung der Mediation Unterschiede zwischen Open Source und kommerziellen Lösungen?

Eine direkte Kommunikation zwischen zwei Anwendungen kann unter anderem durch den Einsatz eines Messaging-Systems (wie WebSphere MQ oder Apache ActiveMQ) in eine indirekte, asynchrone Kommunikation überführt werden. Für den Datenaustausch stellt das Messaging-System dafür einen Container zur Verfügung, auf den beide Applikationen lesend und schreibend zugreifen können. Dieser wird häufig „Queue" genannt. Damit die von der Anwendung „A" in die Queue eingereihte Message von der Anwendung „B" verstanden werden kann, müssen beide sich auf ein einheitliches logisches und physisches Datenformat einigen. Der Abhängigkeitsgrad zwischen den Anwendungen ist also sehr hoch. Es liegt eine enge Kopplung vor. Diese kann im Rahmen einer „Pipes and Filter"-Architektur bei der Kommunikation zwischen den beiden Anwendungen gemildert werden.

Alternative Mediations-Frameworks im Open Source Bereich sind Mule sowie Apache Synapse. Kommerzielle Anbieter betrachten Mediation überwiegend als Teilfunktion ihrer jeweiligen ESBs. Innerhalb der IBM WebSphere Brand existiert beispielsweise aktuell keine rein auf Mediation fokussierte Lösung. Deshalb vergleicht der Artikel Apache Camel mit dem ESB IBM WebSphere Message Broker (nicht zu verwechseln mit dem Active MQ Broker und dem FUSE Message Broker). Dabei kommen nur Funktionen zur Sprache, die beide Produkte unterstützen.

Apache-Camel-Mediationen basieren auf der in dem Buch *Enterprise Integration Patterns* (Gregor Hohpe, Bobby Woolf, Enterprise Integration Patterns: Designing, Building and Deploying Messaging Solutions, Addison-Wesley Longman, 2003) ausführlich dargestellten „Pipes and Filter"-Architektur. Der Weg einer Message von der Quell- zur Ziel-Anwendung wird als „Route" bezeichnet. Diese besteht aus der Verbindung mehrerer in Reihe geschalteter Filter. Derzeit unterstützt Apache Camel 31 der 65 Patterns. Mehrere Routen können in einem Kontext gebündelt werden. Der Kontext wiederum stellt ein Java-Objekt dar, das Methoden anbietet, um die in ihm enthaltenen Java-Routen zu aktivieren. In dem nachfolgenden Vergleich werden die von der jeweiligen Lösung bereitgestellten Filter, der Entwicklungsprozess sowie die Wartbarkeit der fertigen Mediationen Berücksichtigung finden. Grundlage für die Bewertung der beiden Mediationen bildet folgendes fiktives Szenario (siehe Abbildung 1).

Den Einstiegspunkt in die Mediation bildet die Queue „Input Queue". Hier wird eine Nachricht im XML-Format platziert. Diese enthält Informationen zu einem Kunden oder zu einem Kundenauftrag (siehe Tabelle 1).

XML-Dateien mit Kunden-Informationen werden an einen Translator gesendet, der das Format der Message in ein Fixed-

Kunden-Information	Auftrags-Information
<customer> <forename>Max</forename> <surname> Mustermann </surname> <customerid>123456789</customerid> </customer>	<order> <forename>Max</forename> <surname>Mustermann</surname> <customerid>123456789 </customerid> <articleid>12545436</articleid> </order>

Tabelle 1: Struktur der beiden Nachrichten

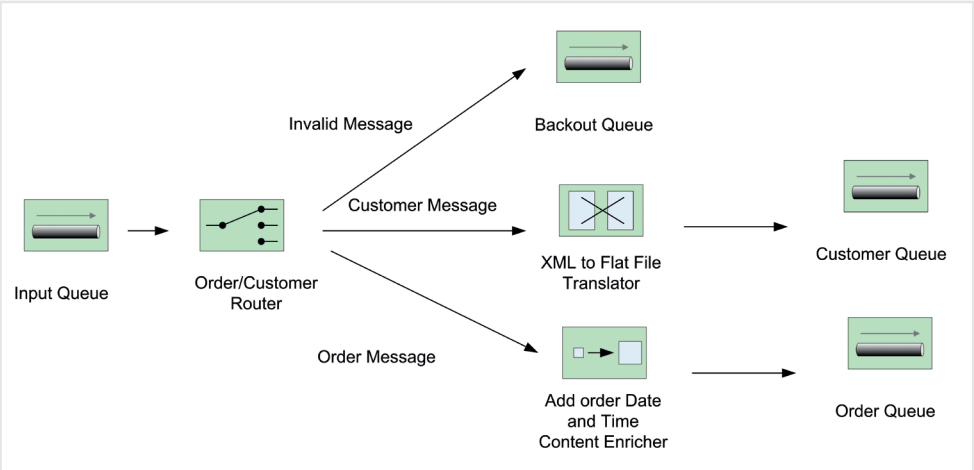


Abbildung 1: Darstellung des Integrationsszenarios

```
Max*******Meier*****123456789*
```

Listing 1: Struktur der resultierenden Fixed-Length-Kunden-Nachricht

```
<order>
  <forename>Max</forename>
  <surname>Mustermann</surname>
  <customerid>123456789</customerid>
  <articleid>12545436</articleid>
  <orderdate>2011-11-30</orderdate>
  <ordertime>13:16:55.647888</ordertime>
</order>
```

Listing 2: Struktur der resultierenden XML-Auftrags-Nachricht

```
CREATE COMPUTE MODULE Conversion_CreateOrderDATA
        CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN
                --automatische generierte Prodzeduraufrufe
                CALL CopyMessageHeaders();
                CALL CopyEntireMessage();
                --diese zwei Anweisungen mussten hinzugefügt werden
                SET OutputRoot.XMLNS.order.orderdate=CURRENT_DATE;
                SET OutputRoot.XMLNS.order.ordertime=CURRENT_TIME;
                RETURN TRUE;
        END;

        --automatisch generierte Prozedur
        CREATE PROCEDURE CopyMessageHeaders() BEGIN
                DECLARE I INTEGER 1;
                DECLARE J INTEGER;
                SET J = CARDINALITY(InputRoot.*[]);
                WHILE I < J DO
                        SET OutputRoot.*[I] = InputRoot.*[I];
                        SET I = I + 1;
                END WHILE;
        END;

        --automatisch generierte Prozedur
        CREATE PROCEDURE CopyEntireMessage() BEGIN
                SET OutputRoot = InputRoot;
        END;
END MODULE;
```

Listing 3

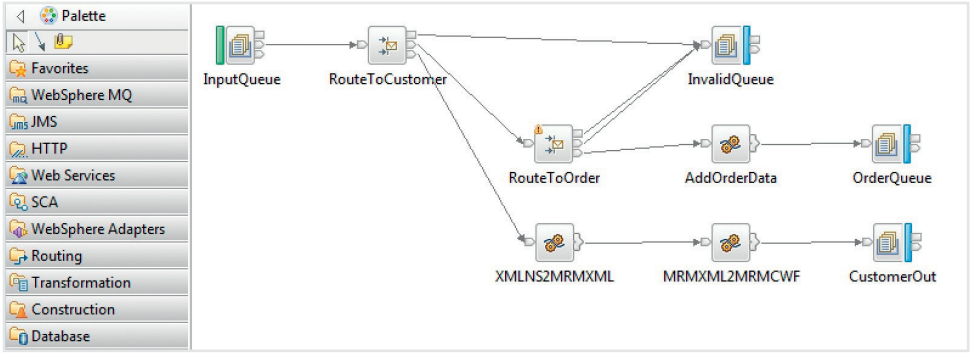


Abbildung 2: IBM WebSphere Message Broker Messageflow

Length-Format ändert. Anschließend wird die Message in die Queue „Customer Queue" eingereiht (siehe Listing 1).

Dagegen werden die XML-Dateien mit Auftragsinformationen an einen Content Enricher gesendet. Dort werden sie um das aktuelle Datum sowie die aktuelle Uhrzeit ergänzt. Anschließend wird die Message in die Queue „Order Queue" eingereiht (siehe Listing 2).

Listing 3 und Listing 4 zeigen eine Gegenüberstellung der Transformationslösung hinsichtlich der Auftragsnachrichten zwischen IBM WebSphere Message Broker und Apache Camel. Die Kundennachrichten sind in Listing 5 und 6 gegenübergestellt.

Adapterumfang von IBM WebSphere Message Broker und Apache Camel

Beide Lösungen stellen zahlreiche Adapter zur Anbindung verschiedener Nachrichten-Formate bereit (wie Web-Services, HTTP, FTP, Dateizugriff, Messaging-Systeme oder Datenbanken). Die einzelnen, innerhalb einer Lösung genutzten Adapter sind in Camel an dem zentralen Einstiegspunkt des Programms definiert. Apache Camel weist diverse Abhängigkeiten zu anderen Projekten auf. Dies spiegelt sich bei der Einbindung der meisten Filter wider – häufig müssen die Dependencies trotz Unterstützung von Apache Maven manuell aufgelöst werden. Dabei kann es passieren, dass im Laufe des Lebenszyklus Dependencies veralten und nicht mehr aufgelöst werden können, was zu Problemen bei Migrationen beziehungsweise Erweiterungen von bestehenden, auf Apache Camel basierenden Lösungen führen kann. Einmal definierte Adapter können als Quelle oder Ziel einer Route genutzt werden. Die Einbindung von Adaptern läuft im WebSphere Message Broker grundlegend anders ab. Aus einer feststehenden Palette können sie als sogenannte „Nodes" innerhalb eines Message-Flow positioniert und anschließend über eine grafische Oberfläche konfiguriert werden. Abhängigkeiten müssen dabei nicht aufgelöst werden. Neben den Protokolladaptern stellt der Message Broker darüber hinaus in einer erweiterten Version auch Applikationsadapter u.a. für SAP- und Peoplesoft-Systeme zur Verfügung.

Zwei sehr häufig innerhalb der Entwicklung einer Mediation genutzte Patterns

sind der Message Router sowie der Message Translator. Der Message Router dient dazu, Nachrichten inhaltsbasiert weiterzuleiten. Der Message Translator wird zur Veränderung des Nachrichteninhalts genutzt.

Message Translator

Insgesamt stellt der WebSphere Message Broker in der aktuellen Version 5 Nodes zur Verfügung, um Transformationen zu implementieren (siehe Abbildung 2). Der ESQL Node eignet sich hervorragend für XML-Manipulationen. Ebenso können Transformationen mittels Java Code oder PHP Code realisiert werden. Auch die Nutzung von XSL wird unterstützt. Mittels Mapping können grafische Transformationen erklickt werden. Dazu muss zuvor die logische und physische Struktur einer Message mithilfe eines Messagesets definiert werden. Die logische Struktur kann auf Basis einer Beispiel-XML-Datei als XSD generiert werden. Dieses muss gegebenenfalls angepasst werden (siehe Listing 7).

Anschließend können für jedes Element der XSD-Datei unterschiedliche physische Repräsentationen definiert werden. Die Eigenschaften der Fixed-Length-Definition des XML-Elements „surname" sind:

- Zehn Zeichen lang
- Leerzeichen werden mit „*" aufgefüllt
- Hat den Wert in /customer/surname
- Feld wiederholt sich nicht

Apache Camel bietet mit der Java-Processor-Klasse lediglich eine Teilmenge der Funktionalität der WebSphere Message Broke Nodes an und ist am ehesten mit dem Java Compute Node vergleichbar (siehe Abbildung 3). Sie bietet eine Schnittstelle an, um den Inhalt einer Message zu lesen, zu manipulieren und anschließend weiterzuleiten. Die Transformation muss selbst in Java implementiert werden. Das mag bei relativ übersichtlichen XML-Dokumenten noch möglich sein – wird allerdings sehr schnell unpraktikabel.

Beinhaltet die XML-Datei mit Auftragsinformationen eine oder mehrere Artikel mit unterschiedlicher ID, müssen die Anzahl der Einträge beispielsweise zur Laufzeit dynamisch durch Xpath-Ausdrücke ermittelt und mithilfe von Kontroll-Struk-

```
public class OrderProcessor implements Processor {

        public void process(Exchange exchange) throws Exception {
                // Auslesen des Inhaltes der Nachricht
                String input = exchange.getIn().getBody(String.class);
                input = input.replaceAll(„\r\n“, „“);
                String output=““;
                Document document=null;
                XMLUtil util = new XMLUtil();

                // XML aus dem String erstellen
                document=util.CreateXMLFromString(input);

                //XML Manipulationen durchführen
        document=util.CreateCustomizedOrderXMLFromXML(document);
                //XML zu String umwandeln
                output=util.CreateStringFromXML(document);
                exchange.getIn().setBody(output);
        }
}
```

Listing 4

```
CREATE COMPUTE MODULE ParseXML
        CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN

                DECLARE inEncoding INT 273;
                DECLARE inCCSID INT 273;

                CALL CopyMessageHeaders();
                -- Parsen der XML Message zu einer Darstellung, die ---mit Messageset
verknüpft werden kann
                DECLARE inBitstream BLOB ASBITSTREAM(InputRoot.
XMLNS,inEncoding,inCCSID);
CREATE LASTCHILD OF OutputRoot DOMAIN(‚MRM‘) PARSE (inBitstream ENCODING in
Encoding CCSID inCCSID SET ‚KC4F64G002001‘ TYPE ‚customer‘ FORMAT ‚CustomerXML‘);

                                --Nutzen des Messagsets für XML Darstellungen
        SET OutputRoot.Properties.MessageSet=‘KC4F64G002001‘;
        SET OutputRoot.Properties.MessageType=‘customer‘;
        SET OutputRoot.Properties.MessageFormat=‘CustomerXML‘;

                RETURN TRUE;
        END;


CREATE COMPUTE MODULE XML2FixedLengthFile
        CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN
                CALL CopyMessageHeaders();
                CALL CopyEntireMessage();

-- Ändern des physischen Typs der Messages von XML
-- auf definiertes Fixed Length File Format
        SET OutputRoot.Properties.MessageFormat=‘CustomerFlatFileCWF‘;

                RETURN TRUE;
        END;
END MODULE;
```

Listing 5

```
public class CustomerProcessor implements Processor {

        public void process(Exchange exchange) throws Exception {
                // Auslesen des Inhaltes der Nachricht
                String input = exchange.getIn().getBody(String.class);
                input = input.replaceAll(„\r\n“, „“);
                String output=““;
                Document document=null;
                XMLUtil util = new XMLUtil();

                // XML aus dem String erstellen
                document=util.CreateXMLFromString(input);

                // Fixed Length File Datei erstellen
        output=util.CreateCustomizedCustomerFlatFileFromXML(document);
                // Veränderte Werte zurückgeben
                exchange.getIn().setBody(output);
        }
}

public String CreateCustomizedCustomerFlatFileFromXML(Document document) throws Excep-
tion{
                String output=““;
                String temp=““;
                int count=0;
//              Vornamen einlesen
                output = document.getFirstChild().getFirstChild().getTextContent();
//              Bis 10.Stelle mit * auffüllen
                count=output.length();
                while (count<10){
                        output=output+“*“;
                        count=count+1;
                }
//              Nachnamen einlesen
                output = output + document.getFirstChild().getFirstChild().getNextSibling().
getTextContent();
//              Bis 20.Stelle mit * auffüllen
                count=output.length();
                while (count<20){
                        output=output+“*“;
                        count=count+1;
                }
//              Kundennummer einlesen
                output = output + document.getFirstChild().getFirstChild().getNextSibling().
getNextSibling().getTextContent();
//              Bis zur 30.Stelle mit * auffüllen
                count=output.length();
                while (count<30){
                        output=output+“*“;
                        count=count+1;
                }
                return output;
        }
```

Listing 6

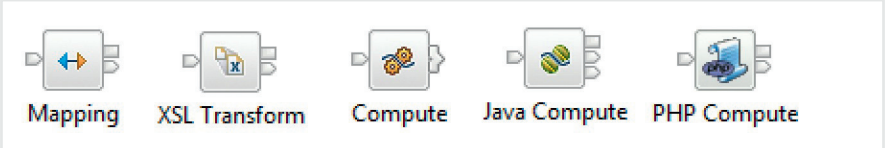


Abbildung 3: WebSphere Message Broker Transformation Nodes

turen alle gefundenen Elemente nacheinander ausgelesen werden. Der WebSphere Message Broker bietet wesentlich mehr Tooling, um mit solchen Freiheitsgraden der XML-Dateien effektiv umgehen zu können.

Message Router

Apache Camel und der IBM WebSphere Message Broker unterstützen beide das Message-Router-Pattern. Für XML-Dateien bieten beide die Auswertung von XML-Element-Inhalten mittels XPath an. Darüber hinaus können anhand der definierten logischen und physischen Datenstruktur im WebSphere Message Broker binär codierte Fixed-Length-Dateien ebenfalls inhaltsbasiert geroutet werden. Durch Anreicherung einer binären Datei mit XML-Meta-Informationen ist dies prinzipiell auch in Apache Camel möglich, erfordert allerdings die Erweiterung einer bestehenden Route um weitere Zwischenschritte (siehe Abbildung 4).

Entwicklungsprozess

Die Basis-Architekturen der beiden Lösungen weisen große Ähnlichkeiten auf. Was bei dem WebSphere Message Broker „Message-Flow“ heißt, kommt der „Route“ in Apache Camel sehr nahe. Ein Message-Flow besteht aus Input-, Verarbeitungs- und Output-Nodes. Ein Node kann als Gegenstück zu den Filtern in Apache Camel gesehen werden. Vereinzelt kann man sogar den Filtern Nodes gegenüberstellen, etwa dem Processor-Filter in Apache Camel den Java Compute Node im WebSphere Message Broker. Trotzdem weichen die Entwicklungsprozesse stark voneinander ab. Apache Camel erfordert umfangreiches Know-how zu Spring, Apache Maven und Java. Transformationen müssen, wie bereits oben erklärt, selbst implementiert, Routen mit XML oder die Java-DSL formuliert und Abhängigkeiten bei Einbindung neuer Filter manuell aufgelöst werden.

Im WebSphere Message Broker erfolgt die Entwicklung größtenteils Tool-gestützt indem die Nodes vereinzelt auf einer Sammelfläche platziert und danach die Properties über eine grafische Oberfläche für diese Nodes gesetzt werden.

Anschließend können Verbindungen zwischen den Nodes gezeichnet werden,

die die Ausführrichtung des Messageflows beschreiben. Bei Manipulationsknoten besteht die Möglichkeit, durch Doppelklick auf den Node nähere Informationen zu den Transformationen einzuholen. Generell basieren diese beim WebSphere Message Broker auf einem Parser, der zwischen logischer und physischer Datenstruktur unterscheidet. Einer logischen Struktur können mehrere physische Datenstrukturen zugewiesen werden. Dabei wird jedem Element in der logischen Struktur eine Darstellung je physischer Struktur zugeordnet. Dadurch kann die physische Struktur einer Nachricht verändert werden, indem lediglich ein anderes physisches Format für die Message gesetzt wird. Soll nur ein Teil der Elemente in das andere physische Format übernommen werden, müssen eine neue logische Struktur und deren zugehörige physische Struktur erstellt werden. Anschließend sind Transformationen zwischen den logischen Datenformaten via Compute (ESQL), JavaCompute (Java) oder Mapping (grafisch) Node möglich. Bei der Erstellung der physischen und logischen Datenformate unterstützen Wizards beziehungsweise grafische Oberflächen.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="forename" type="xsd:string" />
  <xsd:element name="customerid" type="xsd:string" />
  <xsd:element name="surname" type="xsd:string" />
  <xsd:element name="customer">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="forename" />
        <xsd:element ref="surname" />
        <xsd:element ref="customerid" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 7: Aus der Kundennachricht generiertes XSD-Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<order>
     <forename>Max</forename>
     <surname>Meier</surname>
     <customerid>123456789</customerid>
     <articleid>15534</articleid>
</order>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<order>
     <forename>Max</forename>
     <surname>Meier</surname>
     <customerid>123456789</customerid>
     <articleid>15534</articleid>
     <articleid>92424</articleid>
     <articleid>65454</articleid>
     <articleid>25124</articleid>
</order>
```

☑ ☒

Abbildung 4: Grenzen der Java-Processor-Klasse

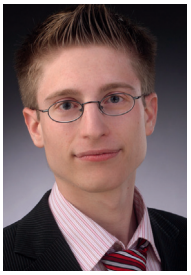
Wartbarkeit

Neben dem Entwicklungsprozess zeigt sich auch im Zuge anfallender Wartungstätigkeiten, dass die Tool-gestützte Entwicklung Vorteile aufweist. Durch die Visualisierung der Schrittfolge von Flows sind diese nahezu intuitiv verständlich. Zudem ist durch Doppelklick auf jeden Node schnell der Zugriff auf den diesem Node zugeordneten Quellcode möglich. Die Flow-Darstellungen bieten weiterhin einen soliden Ansatzpunkt für die Dokumentation der Integrationslösung. Sie können zu bestehenden Flows automatisiert generiert werden. Die Möglichkeit der Wartung von Apache Camel hängt überwiegend von der Sorgfältigkeit des Entwicklers der Integrationslösung ab. Je besser Java-Klassen in verschiedenen Paketen organisiert sind und der Quellcode kommentiert ist, desto leichter wird die Wartung des Quellcodes fallen. Weiterhin wird es bei Nutzung von Apache Camel im Rahmen komplexer Routen schwer fallen, den Überblick über den Verlauf von Routen zu behalten. Die Java-DSL stößt hier schnell an ihre Grenzen.

Fazit

Zusammenfassend kann man sagen, dass Apache Camel als Mediations-Framework gut geeignet ist, allerdings nicht mit einem ESB wie dem WebSphere Message Broker mithalten kann. Apache Camel erleichtert die Anbindung diverser Transportprotokolle enorm und legt eine grundlegende Basis-Architektur für die Realisierung von Integrationslösungen fest. Allerdings müssen die besonders zeitaufwändigen Mediationen weiterhin manuell programmiert werden. Beim WebSphere Message Broker hingegen ist die Basis-Architektur über ein intuitiv verständliches, grafisches Tooling gekapselt. Selbst komplexe Mediationen können über komfortable Bedienoberflächen relativ schnell erstellt werden.

Frank Erbsen
frank.erbsen@x-integrate.com



Frank Erbsen ist Oracle Certified Professional Java SE 6 Programmer und arbeitet derzeit als Developer für Business-Integration-Lösungen bei der X-INTEGRATE Software & Consulting GmbH. Sein Schwerpunkt liegt derzeit auf den ESBs IBM WebSphere Message Broker und Apache ServiceMix.

Unbekannte Kostbarkeiten des SDK
Heute: VisualVM

Bernd Müller, Ostfalia, Hochschule für angewandte Wissenschaften

Das Java SDK enthält eine Reihe von Features, die wenig bekannt sind. Wären sie bekannt und würden sie verwendet, könnten Entwickler viel Arbeit und manchmal sogar zusätzliche Frameworks einsparen. Wir wollen in dieser Reihe derartige Features des SDK vorstellen: die unbekannten Kostbarkeiten.

Das Java SDK enthält bereits seit den allerersten Versionen eine Reihe von Kommandozeilen-Werkzeugen für die Untersuchung von Anwendungen, wie etwa „jps", „jstat", „jstatd", „jinfo", „jhat", „jmap" und „jstack". Mit dem SDK 5 wurde „jconsole" eingeführt, ein Monitoring-Werkzeug, das Informationen über die Performanz und den Ressourcen-Verbrauch von laufenden Java-Anwendungen grafisch darstellt. Mit dem Update 7 des SDK 6 wurde VisualVM eingeführt, nach eigenen Angaben das „All-in-One Java Troubleshooting Tool". VisualVM entstand im Rahmen des NetBeans-Projekts, ist als Stand-alone-Anwendung unter [1] herunterzuladen und – wie bereits erwähnt – in neueren SDKs enthalten.

VisualVM im Überblick

Als Kurzcharakterisierung von VisualVM geben wir die Beschreibung von [1] wieder: „VisualVM is a visual tool integrating several commandline JDK tools and lightweight profiling capabilities. Designed for both production and development time use, it further enhances the capability of monitoring and performance analysis for the Java SE platform."

Im Rahmen dieses Artikels können wir nur einen Überblick über die VisualVM geben und raten dem Leser, nach der Lektüre des Artikels selbst einen Blick auf dieses recht unbekannte, aber nützliche Werkzeug zu werfen. Nach dem Start von VisualVM (Programm „jvisualvm" im „bin"-Verzeichnis des SDK) erscheint das Hauptfenster. Wir verwenden VisualVM in der Version des SDK 6 Update 26.

Man erkennt im linken Teilfenster den Tab „Applications" mit den vier Knoten „Local", „Remote", „VM Coredumps" und „Snapshots". VisualVM erlaubt das Monitoring und Profiling von lokalen und entfernten JVMs. Um letztere mit VisualVM nutzen zu können, müssen einige zusätzliche Vorkehrungen getroffen werden, auf die wir hier jedoch nicht eingehen können. Weiterhin besteht die Möglichkeit, Coredumps zu analysieren sowie Snapshots zu erzeugen und ebenfalls zu analysieren.

Auf der rechten Seite erkennt man im Tab „Start Page" einige Überschriften, die Web-Links darstellen. Wir empfehlen dem Leser zum Einstieg in die SDK-Bordwerkzeuge folgende Links:

- Getting Started with VisualVM
- Troubleshooting Guide for Java SE 6
- Monitoring and Managing Java SE 6

Zurück zu den lokalen Anwendungen. Man erkennt VisualVM selbst, GlassFish, JMeter und Eclipse. Die zu monitorende Anwendung ist durch einen Doppelklick auszuwählen. Im rechten Teilfenster wird dann ein neuer Tab mit Informationen zur ausgewählten Anwendung dargestellt. Der Name des Tabs entspricht dem im linken Teilfenster selektierten Knoten. Der Tab selbst ist wiederum in Unter-Tabs aufgeteilt. Abbildung 1 zeigt den initialen Tab „Overview".

Die Tabs „JVM arguments" und „System properties" sollten die erste Anlaufstelle sein, wenn sich eine Anwendung unerklärlich verhält. Eventuell ist lediglich ein Property falsch definiert.

Monitoring von Anwendungen

Bei Server-Anwendungen kann es zu Speicherplatz-Problemen kommen, die sich durch einen „OutOfMemory"-Error oder häufige Garbage-Collection-Pausen bemerkbar machen. Zur Analyse derartiger Probleme können hochwertige kommerzielle Produkte – für eine erste (und eventuell ausreichende) Analyse aber auch VisualVM eingesetzt werden. In einem kleinen Beispiel wollen wir Objekte auf dem Heap anlegen und wieder freigeben, um den Einsatz des Garbage Collectors zu erzwingen und die Speicherplatz-Belegung und den Einsatz des Garbage Collectors mit VisualVM visualisieren zu können.

Zunächst benötigen wir ein Programm, um Speicherplatz (sinnlos) anzulegen und wieder freizugeben. Wir wählen JAX-RS als Implementierungs-Alternative, da eine Anwendung mit Swing oder ein Menü über „System.in/System.out" aufwändiger wäre. Der folgende Code legt pro Aufruf von „waste()" eine Liste von String-Arrays wachsender Länge an und erzeugt durch eine unglückliche Verwendung von „+" zur String Concatenation zusätzlichen Müll. Da die Liste eine lokale Variable der Methode ist, kann die ganze Struktur nach dem Methodenaufruf „garbage-collected" werden. Die Details der Implementierung sind nicht von Interesse (siehe Listing 1).

Durch die Definition als REST-Service und entsprechende Konfiguration von Anwendungsnamen (waste) und REST-Anwendungspfad (rest) kann mit jedem Browser, mit „wget" oder „curl" und dem folgenden URL der Speicher in Anspruch genommen werden: http://localhost:8080/waste/rest/waste/100. Dabei ist die Größe der Speicherstruktur frei wählbar. Auch der explizite Aufruf des Garbage Collectors kann einfach über REST realisiert werden (siehe Listing 2)

Der Garbage Collector kann nun ebenfalls sehr einfach aufgerufen werden: http://localhost:8080/waste/rest/gc. Für den REST-erfahrenen Leser: Wir sind uns darüber im Klaren, dass „GET" in beiden

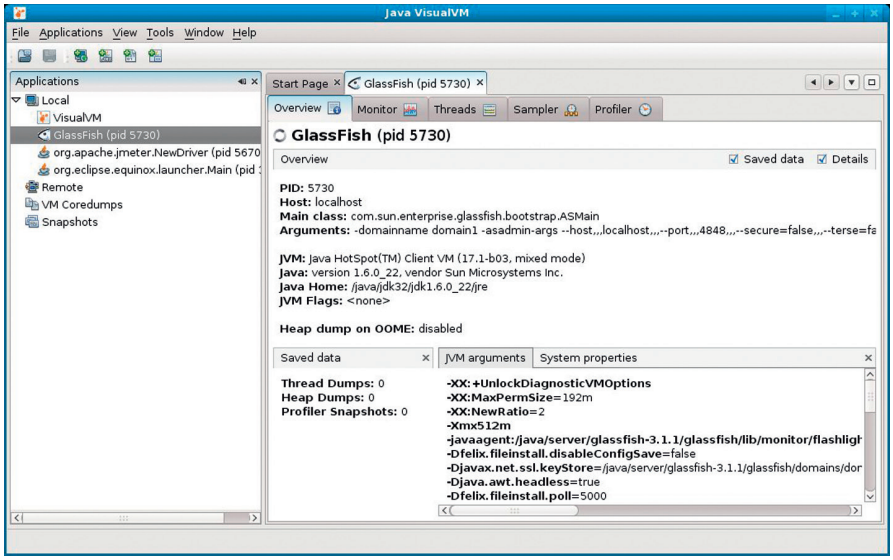


Abbildung 1: Overview mit „JVM arguments“ und „System properties“

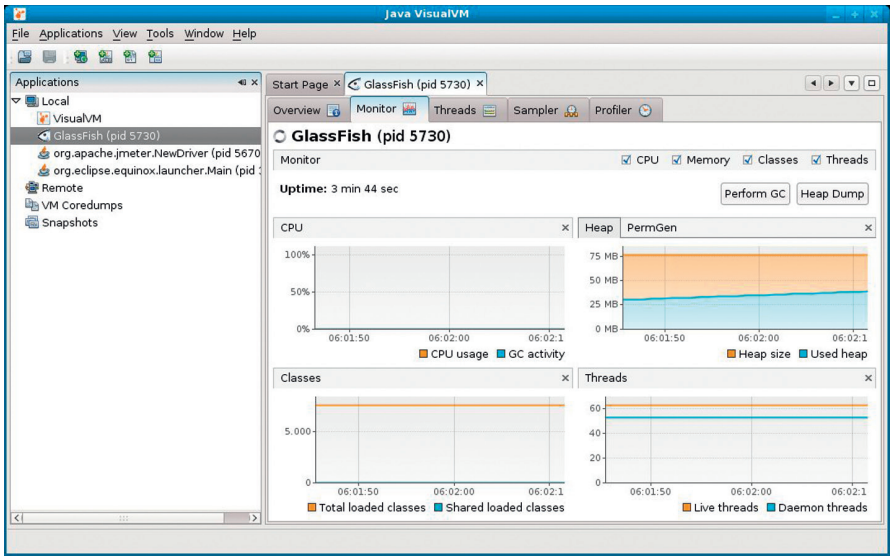


Abbildung 2: Monitor-Tab mit vier Teilbereichen

Fällen nicht die korrekte und von REST intendierte Alternative ist. Sie ist aber am einfachsten zu verwenden.

Zurück zur VisualVM. Unter dem Tab „Monitor“ erscheint eine Anzeige mit den vier Bereichen „CPU“, „Heap/PermGen“, „Classes“ und „Threads“ (siehe Abbildung 2). Da die Darstellung der Klassen und Threads für unser Beispiel irrelevant ist, können diese beiden Teilbereiche entfernt werden. Mit den beiden verbliebenen Bereichen „CPU“ und „Heap“ starten wir nun unseren Versuch. Mit „Jmeter“, einem Werkzeug für Load-Tests und Performanz-Messungen der Apache Software Foundation [2], rufen wir mehrfach die URL „http://localhost:8080/waste/rest/waste/{size}“ auf. Der konkrete Wert von „size“ ist nicht relevant und muss für die jeweilige Umgebung eventuell angepasst werden, um die gewünschte Speichernutzung zu erzwingen. Abbildung 3 zeigt die Darstellung in VisualVM.

Man erkennt, dass der mehrmalige Aufruf der Methode entsprechende Speicher-Allokationen verursacht hat. Um insgesamt mit dem Speicher auskommen zu können, hat die JVM den Garbage Collector mehrfach aufgerufen (Kurve „GC activity“ im linken Diagramm). Die JVM hat außerdem den Heap-Bereich auf 500 MB erweitert, was dem (konfiguriert) maximal verfügbaren Heap entspricht.

Das Plug-in „Visual GC“

Die Reihe „Unbekannte Kostbarkeiten“ stellt SDK-Bordmittel vor. Das Thema „Speichernutzung und Garbage Collection mit VisualVM“ wäre aber ohne das Plug-in „Visual GC“ nicht annähernd vollständig. VisualVM erlaubt durch einen einfachen Plug-in-Mechanismus die Erweiterung um verschiedene Aspekte. Eine Reihe von Plug-ins können von [1] heruntergeladen und installiert werden. Über das Menü „Tools -> Plugins“ geschieht dies über wenige Klicks, sodass durchaus von einem SDK-Bordmittel gesprochen werden kann. Wir gehen außerdem davon aus, dass in einer zukünftigen Version von VisualVM dieses Plug-in bereits enthalten sein wird.

```
@Path(„/“)
public class WasteService {

  @GET
  @Path(„waste/{size}“)
  @Produces(MediaType.TEXT_PLAIN)
  public String waste(@PathParam(„size“) int size) {
    List<String[]> waste = new ArrayList<String[]>();
    for (int i = 1; i < size; i++) {
      waste.add(consume(i));
    }
    return „wasted“;
  }

  private String[] consume(int i) {
    String[] waste = new String[i];
    waste[0] = new String(„1“);
    int digit = 1;
    for (int j = 1; j < waste.length; j++) {
      waste[j] = waste[j-1] + (++digit) % 10;
    }
    return waste;
  }
}
```

Listing 1

```
@Path(„/“)
public class WasteService {
  ...
  @GET
  @Path(„gc“)
  @Produces(MediaType.TEXT_PLAIN)
  public String gc() {
    System.gc();
    return „gc called“;
  }
  ...
}
```

Listing 2

Abbildung 5 zeigt VisualVM nach der Installation von Visual GC und dem geöffneten Tab „Visual GC".

Die Standard-Darstellung des Heap in VisualVM unterscheidet nicht zwischen den verschiedenen Heap-Bereichen (siehe Abbildung 3). Mit Visual GC ist es möglich, diese Bereiche einzeln zu beobachten. In Abbildung 4 ist der Bereich für die Old Generation und die Young Generation zu erkennen. Die Young Generation ist wiederum in die Bereiche „Eden", „Survivor 0" und „Survivor 1" unterteilt. Oracles VM erlaubt unter anderem mit den Startoptionen „-Xmn", „-XX:NewSize", „-XX:MaxNewSize", „-XX:NewRatio" und „-XX:SurvivorRatio" das Fein-Tuning des Heap. Um diese Optionen sinnvoll einsetzen zu können, muss zunächst das Verhalten dieser Speicher-Bereiche auf potenzielle Probleme hin analysiert werden. Die Verwendung von VisualVM ist eine Möglichkeit, dies zu tun.

Fazit

Wir haben die verborgene Kostbarkeit „VisualVM" vorgestellt, die seit Java 6 Update 7 im SDK enthalten ist. Mit ihr ist es möglich, den Speicherbedarf einer Anwendung und das Garbage-Collection-Verhalten einer VM zu analysieren. Leider war nur ein kleiner Einblick in die Features von VisualVM möglich. Nicht vorgestellt haben wir das Profiling. So können etwa für eine Klasse die Anzahl der Instanzen und deren Speicherbedarf sowie die Laufzeit einer Methode ermittelt werden. Durch weitere Plug-ins, etwa für das Monitoring des GlassFish-Application-Servers oder die Anzeige von MBeans-Details, kann VisualVM mit zusätzlicher Funktionalität versehen werden.

Wir raten, VisualVM auszuprobieren und sich ein eigenes Bild von den zur Verfügung stehenden Möglichkeiten zu machen. Für weitere Informationen zu den genannten Themen ist das Buch von Hunt und John empfohlen [3].

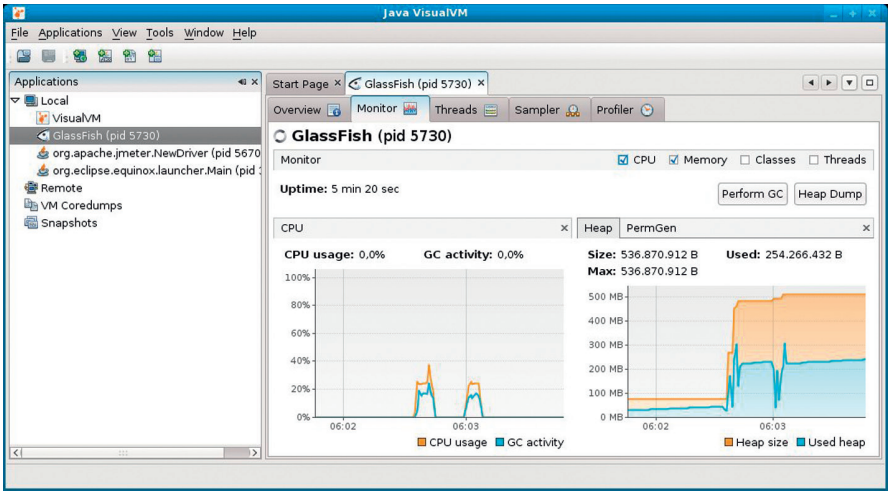


Abbildung 3: CPU- und Heap-Auslastung für den Test

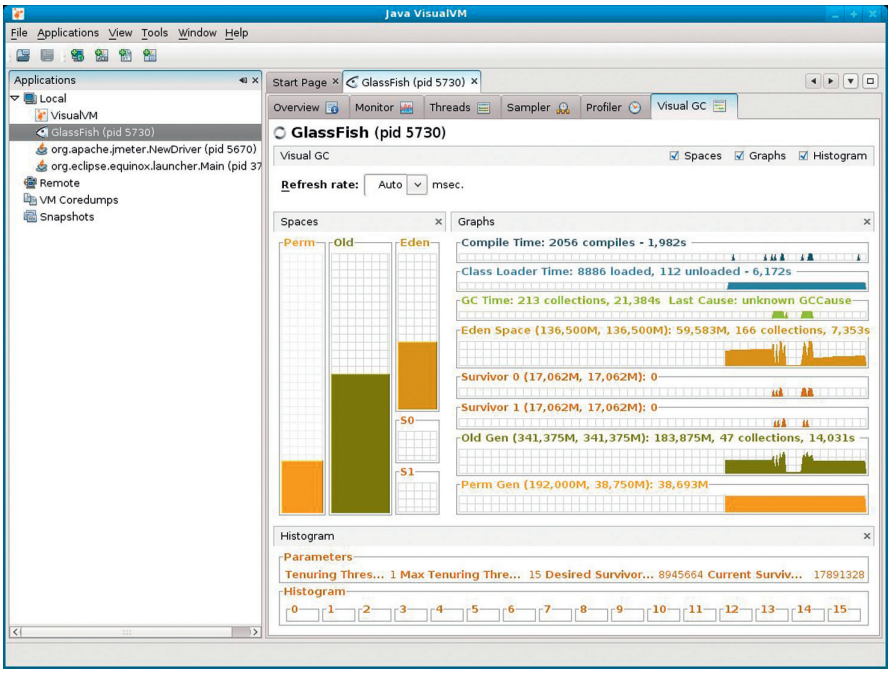


Abbildung 4: Das Tab des Plug-ins Visual GC

Weitere Informationen

[1] http://visualvm.java.net
[2] http://jmeter.apache.org
[3] Charlie Hunt, Binu John. Java Performance. Addison Wesley, 2012

Bernd Müller
bernd.mueller@ostfalia.de

Bernd Müller ist seit März 2005 Professor für Software-Technik an der Fakultät Informatik der Hochschule Braunschweig/Wolfenbüttel, nachdem er sieben Jahre Professor für Wirtschaftsinformatik an der Hochschule Harz war.

Das Eclipse-Modeling-Framework: die API

Jonas Helming und Maximilian Kögel, EclipseSource München GmbH

Mit dem Eclipse Modeling Framework (EMF) können Entitäten einer Anwendung modelliert und als Java-Klassen generiert werden. Dies ermöglicht einen sehr kurzen Entwicklungszyklus vom Datenmodell zu einer lauffähigen ersten Anwendung, in der Entitäten, Attribute und Referenzen bereits in einer Benutzeroberfläche sichtbar sind.

In der letzten Ausgabe haben wir gezeigt, wie Modelle in EMF erzeugt und daraus Code generiert werden kann. Dieser Artikel beschreibt nun den Aufbau des generierten Codes, die API der generierten Klassen sowie nützliche Hilfsklassen. In der nächsten Ausgabe werden dann zwei zusätzliche Frameworks vorgestellt, die im Zusammenspiel mit EMF verwendet werden können und eine zügige Implementierung einer typischen datenzentrischen Anwendung rund um ein EMF-Modell ermöglichen.

In der letzten Ausgabe wurden das Beispiel-Modell in EMF erstellt und daraus Code generiert. Abbildung 1 zeigt dieses Modell, welches das Spiel „Bowling" mit einer League und Tournaments abbildet. Aus diesem Modell wurde bereits Source Code für die Entitäten erstellt. Wer diesen Teil verpasst hat, findet eine englische Version dieses Tutorials unter [1] zum Download. Dort steht auch der Link zum Herunterladen einer Beispiellösung [2]. In der letzten Ausgabe wurde außerdem beschrieben, wie mit einem von EMF generierten Editor bereits per User-Interface Entitäten dieses Modells angelegt werden können.

Dieser erste Test des Modells funktioniert ohne eine einzige Zeile handgeschriebenen Quellcodes. Früher oder später wird man jedoch die generierten Objekte auch programmatisch verwenden. Dazu sind in diesem Artikel die von EMF generierten Klassen vorgestellt, die einem generischen Schema folgen. Es wird gezeigt, wie man Objekte erstellt und modifiziert. Außerdem werden die Möglichkeiten beschrieben, auf Objekte reflexiv zuzugreifen.

Der generierte Code

Werfen wir zunächst einen Blick auf die verschiedenen Plug-ins, die von EMF generiert werden. Bei der Generierung (im Kontextmenü des Wurzelknotens im Generator-Modell) kann man auswählen, ob man nur ein bestimmtes dieser Plug-ins oder alle vier generieren möchte:

- *Modell*
  Das Modell–Plug-in enthält die generierten Entitäten (Java-Klassen), Packages sowie Factory-Klassen, um Instanzen des Modells zu erzeugen. Das Modell-Plug-in ist in der Standard-Einstellung identisch mit dem bereits erzeugten Plug-in, in dem das Modell in einer Ecore-Datei definiert wurde. In diesem Beitrag werden der Aufbau und die Verwendung dieser Klassen detailliert beschrieben.
- *Edit*
  Das Edit-Plug-in enthält zusätzliche Klassen, um Instanzen des Modells in einer Anwendung anzuzeigen, beispielsweise Label Provider, die Icons und Texte für Elemente des Modells liefern.
- *Editor*
  Das Editor-Plug-in ist ein Beispiel-Editor für das generierte Modell. Mit diesem können Instanzen des Modells erzeugt werden, das Modell kann also direkt ausprobiert werden. Er wurde in der letzten Ausgabe vorgestellt.
- *Test*
  Das Test-Plug-in enthält Hüllenklassen für das Erstellen von Test-Cases mit JUnit. In diesem Beitrag werden die generierten Test-Cases verwendet, um die Verwendung der Modell-Klassen zu demonstrieren.

Ausführung der Code-Beispiele

Dieser Artikel erläutert anhand von Code-Beispielen die API von generierten EMF-Objekten und Hilfsklassen. Diese sind durchweg als Testfälle geschrieben. Sie lassen sich damit über JUnit sehr einfach zur Ausführung bringen und testen. Dazu kann beispielsweise das von EMF generierte Test-Plug-in zum Einsatz kommen. Dieses enthält eine Testklasse pro Entität des Modells, beispielsweise „MatchupTest". Die Code-Beispiele können nun in eine beliebige dieser Testklassen eingefügt und über das Kontext-Menü (Run As => JUnit Test)

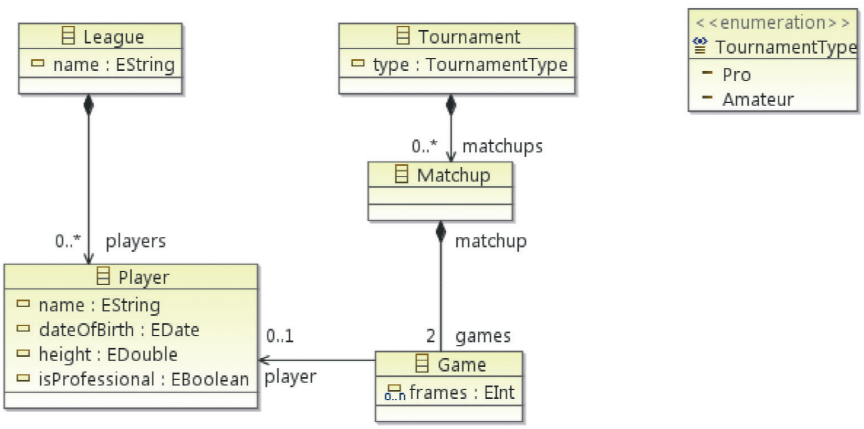


Abbildung 1: Das Beispielmodell

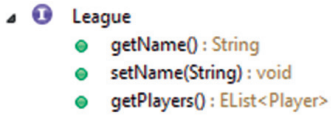


Abbildung 2: Eine Entität mit Getter- und Setter-Methoden

ausgeführt werden. Die Code-Beispiele enthalten zum Ende meist Vergleiche, die überprüfen, ob eine bestimmte Aktion erwartungsgemäß ausgeführt wurde. Diese sind aber nicht wirklich als Testfälle zu verstehen, sondern ausschließlich als Veranschaulichung des Beispiels. Es geht also nicht um ein konkretes Testen der Objekte. Da es sich um generierten Code handelt, wären diese Tests auch nur bedingt sinnvoll.

Die Entitäten

Das Modell-Plug-in enthält die generierten Entitäten, konkret wird für jede Entität ein Interface und eine Implementierung generiert, beispielsweise „League“ und „LeagueImpl“. Die Entitäten enthalten jeweils Getter- und Setter-Methoden für die zuvor modellierten Attribute und Referenzen, die auch als „Features“ bezeichnet werden. Alle generierten Objekte sind in der Standardeinstellung Subtypen der Klasse „Eobject“. Diese implementiert generische Funktionalitäten, auf die wir später noch zurückkommen.

Daneben enthält das Modell-Plug-in „Factories“, um Entitäten zu erzeugen. Da die Konstruktoren von Entitäten üblicherweise private sind, ist dies der vorgesehene Weg, um EObjects zu erzeugen. Da die Factories auch von Frameworks verwendet werden, um beispielsweise Objekte aus Dateien zu laden, sollte man diese nicht verändern. Die Factory, um eine bestimmte Entität zu erzeugen, ist nach dem Package benannt, das die Entität im Modell enthält. Das Beispielmodell enthält nur ein (Wurzel-)Package. Alle Entitäten des Bowlingmodells können damit mit der Factory „BowlingFactory“ erzeugt werden. „Factories“ sind Singletons und können über das Feld „eINSTANCE“ statisch erreicht werden.

Nach dem Erzeugen lassen sich Entitäten über Getter- und Setter-Methoden modifizieren. Referenzen mit einer Multiplizität größer als eins erzeugen dabei nur eine Getter-Methode, die eine Liste zurückgibt. Modifikationen an dieser Liste entsprechen Modifikationen an der Referenz, das heißt die Liste, die man durch den Getter erhält, ist keine Kopie. Listing 1 erzeugt zunächst einen Player und setzt seinen Namen. Anschließend wird überprüft, ob der Name richtig gesetzt wurde. Im zweiten Schritt werden ein zweites Objekt, eine League, erzeugt und der Player als Referenz eingefügt. Das Resultat wird

```
public void testCreateModelelement() {
        Player player = BowlingFactory.eINSTANCE.createPlayer();
        player.setName("Johannes");
        assertEquals("Johannes", player.getName());
        League league = BowlingFactory.eINSTANCE.createLeague();
        league.getPlayers().add(player);
        assertEquals(1, league.getPlayers().size());
}
```

Listing 1

```
public void testEContainer() {
Player player = BowlingFactory.eINSTANCE.createPlayer();
        League league = BowlingFactory.eINSTANCE.createLeague();
        league.getPlayers().add(player);
        assertEquals(league, player.eContainer());
}
```

Listing 2

```
public void testMatchupGameRef() {
        Matchup matchup = BowlingFactory.eINSTANCE.createMatchup();
        Game game = BowlingFactory.eINSTANCE.createGame();
        matchup.getGames().add(game);
        assertEquals(matchup, game.getMatchup());
        assertEquals(matchup, game.eContainer());
}
```

Listing 3

```
public void testIterativeReflection() {
        Player player = BowlingFactory.eINSTANCE.createPlayer();
        player.setName("Johannes");
        for (EAttribute attribute : player.eClass().getEAllAttributes()) {
                player.eUnset(attribute);
        }
        assertEquals(null, player.getName());
}
```

Listing 4

```
public void testReflection() {
        EObject eObject = BowlingFactory.eINSTANCE.createPlayer();
        eObject.eSet(BowlingPackage.eINSTANCE.getPlayer_Name(),
"Johannes");
        Player player = (Player) eObject;
        assertEquals("Johannes", player.getName());
}
```

Listing 5

```
public void testAdapter() {
        Game game = BowlingFactory.eINSTANCE.createGame();
        game.eAdapters().add(new AdapterImpl() {
                @Override
                public void notifyChanged(Notification notification) {
                        super.notifyChanged(notification);
                        notified = true;
                }
        });
        game.setPlayer(BowlingFactory.eINSTANCE.createPlayer());
        assertTrue(notified);
}
```

Listing 6

```
public void testContentAdapter() {
        Matchup matchup = BowlingFactory.eINSTANCE.createMatchup();
        Game game = BowlingFactory.eINSTANCE.createGame();
        matchup.getGames().add(game);
        matchup.eAdapters().add(new EContentAdapter() {
                @Override
                public void notifyChanged(Notification notification) {
                        super.notifyChanged(notification);
                        notified = true;
                }
        });
        assertEquals(1, game.eAdapters().size());
        game.setPlayer(BowlingFactory.eINSTANCE.createPlayer());
        assertTrue(notified);
}
```

Listing 7

```
public void testCopy() {
        Player player = BowlingFactory.eINSTANCE.createPlayer();
        player.setName("Johannes");
        Player copy = EcoreUtil.copy(player);
        assertNotSame(player, copy);
        assertEquals(player.getName(), copy.getName());
}
```

Listing 8

```
public void testValidation() {
        Matchup matchup = BowlingFactory.eINSTANCE.createMatchup();
        matchup.getGames().add(BowlingFactory.eINSTANCE.createGame());
        Diagnostic validate = Diagnostician.INSTANCE.validate(matchup);
        assertEquals(Diagnostic.ERROR, validate.getSeverity());
}
```

Listing 9

über die Anzahl der referenzierten Entitäten überprüft.

Die im vorherigen Beispiel erzeugte Referenz ist unidirektional. Das bedeutet, eine League kennt zwar die enthaltenen Player, ein Player kennt aber, dem Modell nach, seine League nicht. Die Referenz zwischen League und Player wurde jedoch als Containment modelliert. Das bedeutet, jeder Player ist in genau einer League enthalten. EMF erlaubt es daher, generisch auf den Container eines Objekts zuzugreifen. Dazu wird die Methode „eContainer()" aus der Superklasse „Eobjekt" verwendet. Listing 2 veranschaulicht diese Art des Zugriffs.

Soll jedoch generell eine Referenz von beiden Seiten navigierbar sein, sollte sie bidirektional modelliert werden. Dies hat den Vorteil, dass das Gegenüber der Referenz statisch getypt zugreifbar ist. Im Beispielmodell ist die Referenz zwischen Matchup und Game bidirektional modelliert, ein Game kennt also sein Matchup und umgekehrt ein Matchup seine Games. Ausgehend von einem Game kann nun auf ein referenziertes Matchup auf zwei Weisen zugegriffen werden. Entweder getypt über die Methode „getMatchup()" oder generisch über die Methode „eContainer()" (siehe Listing 3).

Reflexiver Zugriff

Neben dem generischen Zugriff über Methoden wie „eContainer()" kann man auf EMF-Entitäten auch reflexiv, das heißt unter Verwendung von Informationen aus dem ursprünglichen Modell zugreifen. Jedes EObject liefert über die Methode „eClass()" ein Objekt vom Typ „Eclass" zurück. Eine EClass beschreibt die vorliegende Entität genauer hinsichtlich des Modells als die Class-Klasse in Java. Mithilfe der EClass lässt sich zum Beispiel die maximale Multiplizität einer Referenz bestimmen.

Eine EClass enthält eine Liste aller Attribute einer vorliegenden Entität. Diese Liste besteht aus Instanzen vom Typ „Eattribut", die jeweils ein bestimmtes Attribut beschreiben. Das Attribut kann nun auch verwendet werden, um es reflexiv an der Entität zu modifizieren. Dazu können die Methoden „eSet", „eGet" und „eUnset" verwendet werden, die jeweils ein Feature (Attribut oder Referenz) als Parameter erwarten. eUnset setzt dabei ein Feature auf den Ursprungswert zurück. Listing 4 erzeugt einen Player und setzt ein Attribut. Anschließend werden reflexiv alle Attribute, inklusive der Attribute von Superklassen des Players, auf den Ausgangswert zurückgesetzt und das Ergebnis überprüft.

Will man nicht auf eine Liste von Attributen, sondern auf ein ganz bestimmtes Attribut zugreifen, kann man dieses statisch aus dem generierten Package abfra-

gen. Ein Package befindet sich – ebenso wie die Factories für jedes Package aus dem Modell – im Modell-Plug-in. Listing 5 zeigt, wie ein bestimmtes Attribut, im Beispiel das Name-„Attribut" der Klasse „Player" über die reflexive Methode gesetzt und anschließend über die entsprechende statische Methode überprüft wird. Ein derartiger Zugriff ist natürlich nur begrenzt sinnvoll. Es wird davon ausgegangen, dass das vorliegende Objekt ein Player ist; in diesem Fall könnte es auch direkt gecastet werden. Der direkte Zugriff auf Attribute und Referenzen aus dem Package kann aber beispielsweise zum Auffinden eines bestimmten Attributs aus einer Liste verwendet werden.

Notifizierungen

Einer der sichtbarsten Vorteile von EMF ist der Notifizierungs-Mechanismus der Entitäten. Über diesen kann man sich programmatisch an EObjects registrieren, um über Änderungen notifiziert zu werden. So kann beispielsweise ein Benutzer-Interface auf eine Änderung im Modell reagieren. Sollen Oberflächen-Elemente direkt an einzelne Attribute oder Referenzen gebunden werden, beispielsweise ein Textfeld als String-Attribut, sollte man das Framework „Databinding" einsetzen [3].

Die der Notifizierung zugrunde liegende Logik sowie die entsprechende API werden komplett von EMF generiert. Um auf Änderungen auf einer bestimmten Entität zu reagieren, muss ein Adapter registriert sein. Für einfache Implementierungen kann von der Klasse „AdapterImpl" geerbt werden. Lediglich die Methode „notifyChanged" muss dann überschrieben werden. Diese wird aufgerufen, sobald das EObject modifiziert wird. Die übergebene Notification enthält Informationen darüber, welches Feature auf welche Art und Weise verändert wurde. Listing 6 registriert einen Adapter auf einem Game, modifiziert eine Referenz und überprüft, ob der Adapter korrekt notifiziert wurde.

In einigen Fällen kann es sinnvoll sein, einen Adapter nicht nur auf eine einzelne Entität zu registrieren, sondern auf einer bestimmten Menge der Entitäten. Im Beispiel könnte man etwa an allen Änderungen innerhalb eines Matchup, also auch an Änderungen der enthaltenen Games interessiert sein. Damit nicht manuell an jedem EObject ein Adapter registriert werden muss, gibt es die Hilfsklasse „EcontentAdapter". Diese kann an nur einem Objekt manuell registriert werden und registriert (und deregistriert) sich automatisch rekursiv an allen Kindern des Objekts. Die Registrierung wird auch automatisch aktualisiert, wenn beispielsweise ein neues Element in den Baum eingefügt wird. Listing 7 zeigt einen EContentAdapter, der wie gerade beschrieben auf einem Matchup registriert wird. Anschließend wird sowohl überprüft, dass der Adapter auch auf dem Game hängt, als auch, dass er auf eine Änderung am Game notifiziert wird.

Weitere Hilfsklassen

Neben dem Notifizierungs-Mechanismus stellt EMF zahlreiche weitere Hilfsklassen zur Verfügung, beispielsweise die Klasse „EcoreUtil". Generell steht für die meisten Aufgaben, die in generischer Weise auf EObjekten erfüllt werden können, Framework-Unterstützung zur Verfügung. Es lohnt sich also meist etwas zu suchen, bevor man etwas selber implementiert. Eine gute Anlaufstelle dafür ist die EMF-Newsgroup, in der nur wenige Fragen noch nicht gestellt wurden und außerdem keine Frage unbeantwortet (durch Ed Merks) bleibt [7].

Mit dem EcoreUtil können beispielsweise Kopien von Entitäten erstellt werden. Die Kopien sind dabei im Standardfall „self-contained", das bedeutet, es werden alle Attribute, alle als Containment referenzierten Objekte, aber keine Cross-Referenzen kopiert. Listing 8 kopiert einen Player und überprüft anhand eines Attributs die Kopie.

Eine weitere Unterstützung, die EMF bietet, ist die Validierung von Entitäten. Im einfachsten Fall werden Regeln validiert, die im Modell abgebildet wurden, beispielsweise die Multiplizität einer Referenz. Im Beispiel-Modell besitzt die Referenz von Matchup auf Game die Multiplizität „zwei", es müssen also immer genau zwei Games in einem Matchup enthalten sein. Dies wurde im Modell ausgedrückt, indem sowohl der „Lower Bound" als auch der „Upper Bound" auf „zwei" gesetzt wurden. Solche Einschränkungen finden sich in EMF nicht direkt in der API wieder. Ansonsten wären zur Laufzeit keine invaliden Zwischen-Zustände möglich. Im Beispiel könnte man nicht zuerst ein, dann ein zweites Game in ein Matchup einfügen, weil der Zwischen-Zustand verhindert würde. Stattdessen können Einschränkungen dieser Art in EMF zu bestimmten Zeitpunkten überprüft werden. Dies lässt sich mit einer Hilfsklasse bewerkstelligen (siehe Listing 9). Es zeigt die manuelle Validierung eines Matchup. Da dieses nur ein Game enthält, gibt die Validierung erwartungsgemäß einen Fehler zurück.

Zusätzlich zu den modellierten Einschränkungen lassen sich Validierungsregeln auch im Code formulieren. Siehe [4] für weitere Informationen.

Fazit

Dieser Teil der Reihe „Das Eclipse Modeling Framework" beschrieb die Verwendung der API der von EMF generierten Objekte, beispielsweise um neue Instanzen des Modells programmatisch zu erzeugen und zu modifizieren. Wir stellten Hilfsklassen vor, die generische Aufgaben wie beispielsweise das Kopieren von Objekten erleichtern. In den kommende Teilen der Reihe beschreiben wir zusätzliche Technologien, die mit einem EMF-Modell verwendet werden können, zum Beispiel eine Serverlösung für EMF-Modell-Instanzen. Sollten Sie Fragen zum Artikel oder zu EMF im Allgemeinen haben, kontaktieren Sie gerne die Autoren unter [5]. Weiterhin bieten die Autoren auch Schulungen zum Thema EMF an [6].

Links

[1] http://eclipsesource.com/emftutorial
[2] http://eclipsesource.com/blogs/wp-content/uploads/2011/03/exampleSolution.zip
[3] http://eclipse.org/databinding
[4] http://eclipse.org/emfclient/documentation/validation.php
[5] jhelming@eclipsesource.com
[6] http://eclipsesource.com/en/services/eclipse-training/eclipse-modeling/
[7] http://www.eclipse.org/modeling/emf/newsgroup-mailing-list.php

Jonas Helming
jhelming@eclipsesource.com
Maximilian Kögel
mkoegel@ eclipsesource.com

Jonas Helming und Maximilian Kögel sind Eclipse-EMF/RCP-Trainer und Consultants sowie Geschäftsführer der EclipseSource München GmbH. Sie leiten die Eclipse-Projekte „EMFStore" und „EMF Client Platform".

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift *DOAG News* und vier Ausgaben im Jahr *Business News* zusammen für 75 EUR. Weitere Informationen unter www.doag.org/go/shop

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN
0700 11 36 24 39

ODER BESTELLEN SIE ONLINE
go.ijug.eu/go/abo

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

☐ **Ja**, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr
☐ **Ja**, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:
Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Wiederrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.